UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Gian Ricardo Berkenbrock

Uma ferramenta para o desenvolvimento de modelos de simulação integrada ao ambiente *grid*

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Paulo José de Freitas Filho, Dr.
Orientador

Florianópolis, março de 2005

Uma ferramenta para o desenvolvimento de modelos de simulação integrada ao ambiente *grid*

Gian Ricardo Berkenbrock

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Raul Sidnei Wazlawick, Dr.

Coordenador do Curso

Banca Examinadora

Paulo José de Freitas Filho, Dr.

Orientador

Ricardo José Rabelo, Dr.

Luis Fernando Friedrich, Dr.

Mário Antonio Ribeiro Dantas, Dr.

"Não há maior sinal de loucura do que fazer a mesma coisa repetidamente e esperar a cada vez um resultado diferente."
Albert Einstein

Dedico esta dissertação à minha querida mãe, Rita (em memória), ao meu pai, Nelson, e à minha amada esposa, Carla Diacui.

Agradecimentos

A meu pai, que sempre me apoiou para atingir meus sonhos e que esteve sempre presente para me ajudar.

Ao orientador, Paulo José de Freitas Filho, por acreditar nas minhas idéias, pela amizade e conhecimentos transmitidos durante este período de convivência.

À minha esposa, Carla, pela ajuda, paciência, amor e dedicação.

Aos com meus irmãos, Guy e Guio, que me apoiaram.

Ao pessoal do laboratório PerformanceLab, que contribuíram na conclusão deste trabalho: Eduardo, Fernando (Capin), Rogério, Amândio, Graziela, Rivalino. Valeu!

Aos integrantes da "cúpula": Thiago, Henrique, Odilon, Rafael, Alexandre, Katiane e Rosana, que mesmo distantes me ajudaram. Valeu, galera!

Ao colegas do mestrado, principalmente Leonardo, Alex, Rista, Carlos (Goiano), Marcelo e Dennis.

Ao colega Alexandre Parra, pela ajuda na montagem do ambiente de teste no PerformanceLab.

A todos que de alguma forma ajudaram nesta etapa.

À Capes, pelo apoio financeiro parcial.

A Deus, pela oportunidade.

Sumário

Lista de Figuras

Lista de Quadros

Lista de Siglas

SO	Sistema Operacional
MPP	*Massively Parallel Processors*
COTS	*Commodity Off The Shelf*
MPI	*Message–Passing Interface*
PVM	*Parallel Virtual Machine*
DSM	*Distributed Shared Memory*
OV	Organização Virtual
DEVS	*Discrete Event System Specification*
PL	Processo Lógico
SRIP	*Single Replication In Parallel*
MRIP	*Multiple Replication In Parallel*
XML	*Extensible Markup Language*
DTD	*Document Type Definitions*
UML	*Unified Modeling Language*
FIFO	*First In First Out*
CoG	*Commodity Grid*
CTime	*Current Time*

Resumo

Com o atual aumento da capacidade computacional, pode-se montar ambientes de alto desempenho com computadores de baixo custo. Dessa forma, esses ambientes têm se tornado cada vez mais populares. Existem vários tipos de configurações que podem prover o alto desempenho. No entanto, há baixa oferta de ambientes de modelagem integrados com *grid* para o público-alvo (modeladores). Nesta dissertação foi implementada uma ferramenta que possibilita a criação de modelos de simulação discreta e a execução destes em um *grid* computacional. Essa ferramenta é composta de duas partes: editor e processador de modelos. O editor pode criar e alterar modelos de simulação discreta. O processador foi desenvolvido para interpretá-los e executá-los. As partes se comunicam por meio de um arquivo que contém a representação do modelo, o qual está descrito em XML. Alguns testes foram realizados com o intuito de validar as estatísticas geradas pelo processador de modelos. Esses testes foram satisfatórios e mostram a possibilidade da integração das ferramentas de construção de modelos com os ambientes em *grid*.

Palavras-chaves: Simulação discreta, Modelos discretos, Computação em *grid*, e Simulação paralela e distribuída.

Abstract

With the current growth of computational capacity, it is possible to build high performance computing facilities using low-cost computers. Because of this possibility, these facilities are becoming more popular. There are many different computer configurations that can provide high performance computing. However, for system analysts, there are very few types of modeling software that are integrated with a grid computing facility. Software was developed in this thesis which can create and update discrete simulation models and run them using a grid. This software has two parts: an editor and a model processor. The editor can build and update the models. The processor can read and process the models. The parts communicate using a file which contains the model in XML. Tests were developed to validate the model processor's output. The test results were satisfactory, showing the possibility of connecting a tool that can build discrete models to a grid computing facility.

Keywords: Discrete Simulation, Discrete Models, Grid Computing, and Parallel and Distributed Simulation.

Capítulo 1

Introdução

1.1 Introdução geral

Com o atual aumento da capacidade computacional, ambientes de alto desempenho estão tornando-se cada vez mais populares [PAT 03, SUL 03]. Eles podem ser montados com computadores de baixo custo, o que os torna populares, e são capazes de executar programas seqüenciais ou paralelos. Para executar os programas paralelos, existem conjuntos de programas/bibliotecas que provêm a infra-estrutura necessária.

Há vários tipos de configuração que podem prover o alto desempenho, um deles é o *cluster* de computadores. Os computadores do *cluster* funcionam como se fossem um único sistema para o usuário, compondo um ambiente de alta disponibilidade, isto é, caso ocorra alguma falha em um dos computadores que o compõem, o usuário final não a perceberá. Outra configuração abordada nesta dissertação é a da computação em *grid*. Este tipo de computação surgiu na década de 90. A computação em *grid*, conforme mencionado em Dantas [DAN 02], é definida como computação paralela geograficamente distribuída.

Para utilizar melhor esses ambientes, é necessário o desenvolvimento de aplicações próprias para esse fim. Um exemplo dos tipos de programas que necessitam desse poder computacional são os programas voltados a executar modelos de simulação discreta no menor tempo possível. Para atingir esse objetivo, existe a abordagem para

executar os modelos de forma paralela ou distribuída.

Pode-se ressaltar duas vantagens em se executar um modelo em paralelo ou distribuído sobre a execução em modo seqüencial: (1) tempo de execução reduzido e (2) tolerância à falha em tempo de execução. Cabe destacar que para esse processamento dos programas em paralelo se faz uso de protocolos de sincronização. Esses são responsáveis por executar os eventos na ordem correta.

Contudo, esses programas precisam de um ambiente de alto desempenho para serem executados. Nesta dissertação é enfatizada a utilização de *grids* computacionais, por se tratar de uma tecnologia que está em desenvolvimento. Existem vários projetos de desenvolvimento de um *middleware* para *grid*. Nesta dissertação utilizou-se o *globus toolkit*.

Há vários programas para a construção de modelos de simulação discreta, porém são poucos os que podem ser utilizados para criar modelos com a opção de executá-los em um *grid*. Na atualidade, se um usuário desejar construir um modelo de simulação para ser executado no *grid*, ele precisa ser conhecedor não apenas da área em questão para a construção do modelo mas também de lógica e linguagem de programação.

1.2 Justificativa

Uma ferramenta que possa criar modelos e os enviar para a execução em um ambiente de alto desempenho pode ajudar a ganhar produtividade dos modeladores e melhorar a utilização dos recursos de uma determinada organização. No entanto, existe baixa oferta de ambientes de modelagem integrados com *grid* para o público-alvo (modeladores). Como conseqüência, a execução de modelos de simulação fica restrita a usuários com conhecimento muito específico. Dessa forma, não permite que o modelador se preocupe apenas com o mais importante, que é a modelagem.

1.3 Objetivos

1.3.1 Objetivo geral

Desenvolver uma ferramenta gráfica voltada à criação de modelos de simulação discreta e à execução dos modelos em *grid* computacional.

1.3.2 Objetivos específicos

Esta dissertação tem como objetivos específicos:

a) pesquisar meios para representar modelos de simulação discreta;

b) estudar as bibliotecas existentes para o desenvolvimento de ferramentas gráficas;

c) pesquisar os ambientes de alto desempenho;

d) estudar o processamento dos modelos;

e) pesquisar formas de integração com o ambiente *grid*; e

f) implementar e validar a ferramenta proposta.

1.4 Estrutura do documento

Este documento é composto de seis capítulos: Introdução, Computação de alto desempenho, Simulação, Editor visual integrado ao ambiente *grid*, Resultados experimentais e Conclusão.

O primeiro capítulo apresenta a introdução da dissertação. No segundo apresentam-se conceitos da computação de alto desempenho e, no seguinte, conceitos da simulação. O quarto capítulo trata da especificação e apresentação do programa desenvolvido para a criação de modelos discretos para simulação integrada ao ambiente *grid*. No quinto capítulo alguns resultados experimentais são apresentados e comentados. No sexto e último capítulo são delineadas as conclusões sobre a pesquisa desenvolvida bem como comentários e propostas de trabalhos futuros.

Capítulo 2

Computação de alto desempenho

Atualmente tem aumentado a demanda por poder computacional para os programas que tenham a finalidade de resolver problemas críticos/complexos e obter suas respostas no menor tempo possível [PIT 03, PAT 03, SUL 03]. Além disso, freqüentemente, os programas necessitam de mais poder computacional do que um computador seqüencial pode prover [BUY 99]. No entanto, conforme Maccabe [MAC 93], nem sempre é óbvio como milhares de processadores podem resolver um dado problema.

Para tentar suprir essas necessidades, tem ocorrido, já há algumas décadas, o desenvolvimento das máquinas paralelas. Essa área, entretanto, demonstrou claramente seus fatores críticos, como largura de banda, latência das redes e gerenciamento [STE 01, BUY 99].

Segundo os autores Meredith [MER 03], Apon [APO 01], Van Der Steen [vdS 03], Vaughan [VAU 03] e Buyya [BUY 99], o custo de desenvolvimentos de máquinas paralelas é relativamente alto. Por conta desses custos, surgiu então uma opção mais barata e, teoricamente, tão potente quanto as máquinas paralelas, denominada *cluster* (agregado) de computadores, tema abordado na seção 2.1. Na linha de evolução do desenvolvimento da computação de alto desempenho surgiu, em meados dos anos 1990, o conceito de *grid*, que é uma proposta de infra-estrutura de computação distribuída para a ciência e a engenharia avançada [FOS 99] (ver mais adiante, na seção 2.2).

2.1 Computação em *cluster*

2.1.1 Definição

Um *cluster*, ou agregado, é um conjunto de computadores que são conectados através de uma rede de interconexão de alta ou baixa latência, trabalhando cooperativamente para resolver um determinado problema/tarefa. Dessa maneira, o usuário tem a noção de que vários computadores são apenas um [PIT 03, PAT 03, MAC 93, BUY 99]. Atualmente, o *cluster* tem se tornado uma opção de uso para as empresas, universidades e centros de pesquisa [De 04]. Uma das razões no aumento do uso dos *clusters* se dá pela não-obrigatoriedade de se usarem computadores de um único fornecedor. Podem ser construídos usando-se os computadores comumente denominados *Commodity Off The Shelf* (COTS), que são computadores de baixo custo encontrados facilmente em lojas [STE 01, PIT 03, APO 01]. Esses computadores podem ser tanto multiprocessados quanto com apenas um processador.

Segundo Buyya [BUY 99], a sua arquitetura típica é composta de diversos componentes (figura 2.1), que estão descritos abaixo:

a) múltiplos computadores de alto desempenho;

b) sistemas operacionais;

c) redes de alto desempenho;

d) placas de rede;

e) serviços e protocolos de comunicação rápida;

f) *cluster middleware*:

- escalonadores;
- outros componentes;

g) ferramentas e ambientes de programação paralela:

- memória compartilhada distribuída;
- trocas de mensagens (MPI e PVM); e

h) aplicações:

- seqüenciais;
- paralelas.

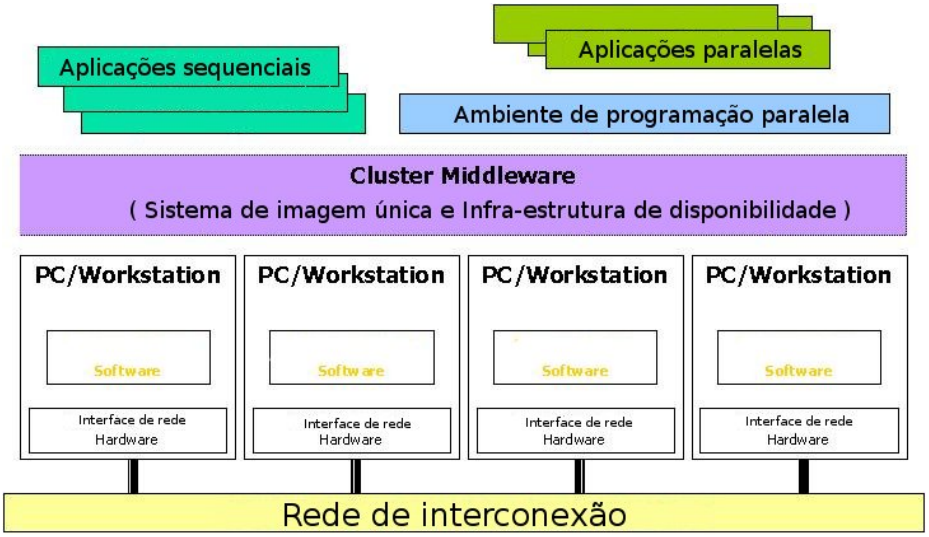

Figura 2.1: Arquitetura de um *Cluster* [BUY 99]

Para melhor esclarecer a computação em *cluster*, é de grande valia uma abordagem na classificação de arquiteturas paralelas. Para tanto, temos a taxonomia de Flynn [FLY 72] e sua extensão [SKI 88], e uma classificação de máquinas paralelas, de acordo com o compartilhamento de memória.

2.1.2 Classificação de arquiteturas paralelas

O uso da classificação em uma determinada área ajuda pesquisadores, estudiosos ou usuários a entender as principais características de um determinado objeto de estudo. Taxonomias são importantes para classificar o mundo. Boas taxonomias devem agrupar aqueles objetos que são fortemente relacionados de um modo importante,

como, por exemplo, mamíferos [SKI 88]. Para a classificação de arquiteturas paralelas, Flynn [FLY 72] propôs a seguinte taxonomia, resumida no quadro 2.1. Ele classificou as arquiteturas pelo número de fluxos de instruções e dados que podem ser processados simultaneamente. Em 1988, Skillicorn [SKI 88] propôs uma extensão no que se refere à taxonomia de diferentes arquiteturas de multiprocessadores. Outra extensão dessa classificação, mais conhecida, é a de Feng [FEN 72], em que é apresentado um esquema de classificação orientado a desempenho. Existe também uma classificação de acordo com o compartilhamento de memórias em multiprocessadores e multicomputadores que pode ser consultada em Pitanga [PIT 03], Maccabe [MAC 93] e De Rose [De 04].

Fluxo de dados/Fluxo de instruções	SI (*Single Instruction*)	MI (*Multiple Instructions*)
SD (*Single Data*)	SISD	MISD
MD (*Multiple Data*)	SIMD	MIMD

Quadro 2.1: Resumo da taxonomia de Flynn

A seguir é realizada uma comparação entre os principais representantes da computação em *cluster*: os processadores massivamente paralelos e os agregados de computadores.

2.1.3 Comparação entre MPP e *Cluster*

Para realizar esta comparação, é necessário, primeiro, definir o que é um *Massively Parallel Processors* (MPP). Os MPP são grandes sistemas de processamento paralelo com a arquitetura de memória compartilhada e centralizada. A interligação entre os processadores ocorre por meio de uma rede de alta velocidade e baixa latência [SUL 03, PIT 03, De 04]. Os *clusters* foram definidos na seção 2.1.1. A partir da proposta de alguns autores [PAT 03, PIT 03, De 04, SUL 03], pode-se montar um quadro comparativo (quadro 2.2).

	MPP	*Cluster*
Rede	Alta Velocidade (Não-padronizada)	Baixa/Alta Velocidade (Padronizada)
Custo de gerenciamento	Baixo	Alto
Disco local nos nós	Ausente	Opcional (Ausente ou não)
SO nos nós	Apenas uma parte é necessária (*kernel*)	Exige a presença do SO
Acoplamento	Forte	Fraco
Ambientes de programação	DSM, MPI	MPI, DSM
Custo de aquisição	Alto	Baixo

Quadro 2.2: Comparação entre MPP e *Cluster*

2.1.4 Tipos de *cluster*

Os *clusters* podem ser classificados de acordo com algumas categorias, dependendo do critério que é escolhido [BUY 99]. Essas categorias podem ser as seguintes.

1. Por aplicação
    - Alto desempenho
    - Alta disponibilidade
2. Por propriedade
    - Dedicado
    - Não dedicado
3. Hardware do nó
    - *Clusters* de PCs (CoPs)
    - *Clusters* de estações de trabalho (COWs)
    - *Clusters* de SMPs (CLUMPs)

4. Sistema operacional do nó
    - Linux *Clusters*
    - Solaris *Clusters*
    - AIX *Clusters*
5. Configurações dos nós
    - Homogêneo
    - Heterogêneo
6. Níveis de agrupamento
    - Agregados em grupo (#nós:2–99): usa *System Area Networks* (SANs) como *Myrinet*
    - Agregados Departamentais (#nós:10–100)
    - Agregados Organizacionais (conjunto de agregado departamental)
    - Metacomputador Nacional (WAN/Internet-based) (conjunto dos dois anteriores)
    - Metacomputadores Internacionais (Internet-based) (de mil a milhões de nós)

2.2 Computação em *grid*

A computação em *grid* surgiu em meados dos anos 90 [FOS 99], com uma nova visão de computação distribuída. Ela tem foco no compartilhamento de recursos em larga escala, inovações em programas e em computação de alto desempenho [FOS 01].

2.2.1 Definição

A computação em *grid*, conforme mencionado em Dantas [DAN 02], pode ser definida como computação paralela geograficamente distribuída. Esse conceito

completa a definição de Foster [FOS 99], segundo o qual um *grid* computacional é definido como uma infra-estrutura de computadores e programas que provêem um acesso confiável, coerente, difundido e econômico, com potencial computacional final. Completando, Foster [FOS 01] comenta que a computação em *grid* está preocupada com o compartilhamento de recursos coordenados e comprometida com problemas em organizações virtuais (OV) dinâmicas e multi-institucionais (ilustrado na figura 2.4). Uma OV é um conjunto de indivíduos e/ou instituições definido pelas regras de compartilhamento dos recursos existentes no *grid* [FOS 01].

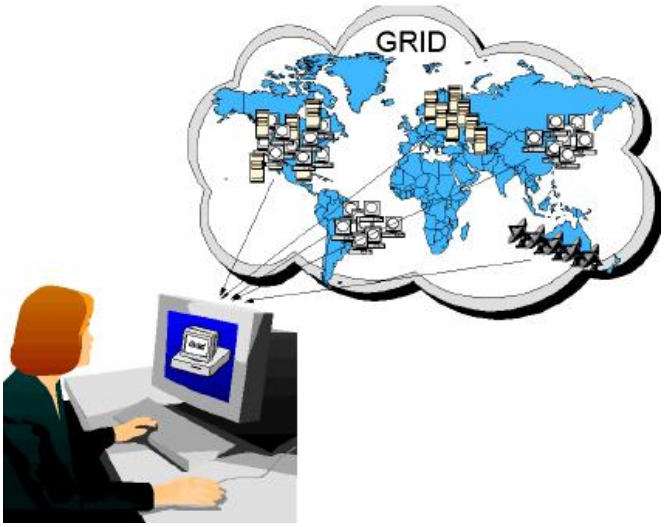

Figura 2.2: Exemplo de um *grid* [FER 03]

Para melhor compreender esses conceitos, pode-se apresentar a analogia com as distribuidoras de energia. Estas possuem uma rede de fornecimento de energia elétrica, a qual tenta manter transparente para o usuário de onde vem a energia. O usuário apenas se preocupa em consumir energia. Problemas de distribuição e geração, na maioria das vezes, não são percebidos pelo usuário final [FOS 99, JAC 03]. Para a computação em *grid* atingir esse grau de transparência, a padronização é a chave. Padronizando arquiteturas, isto é, desenvolvendo especificações abertas ao público, será aumentada a sinergia para se atingir o objetivo. Tentando atingir esse e outros objetivos é que o *Global Grid Forum* (GGF) [FOR 03] desenvolveu a *Open Grid Services Architecture* (OGSA). A OGSA

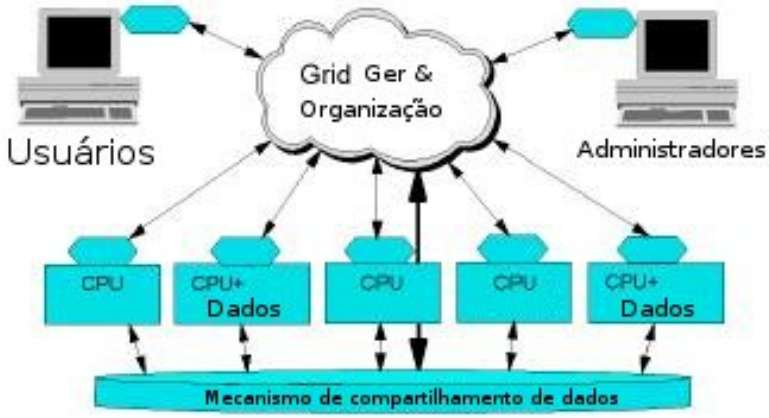

Figura 2.3: Exemplo de um *grid* simples [FER 03]

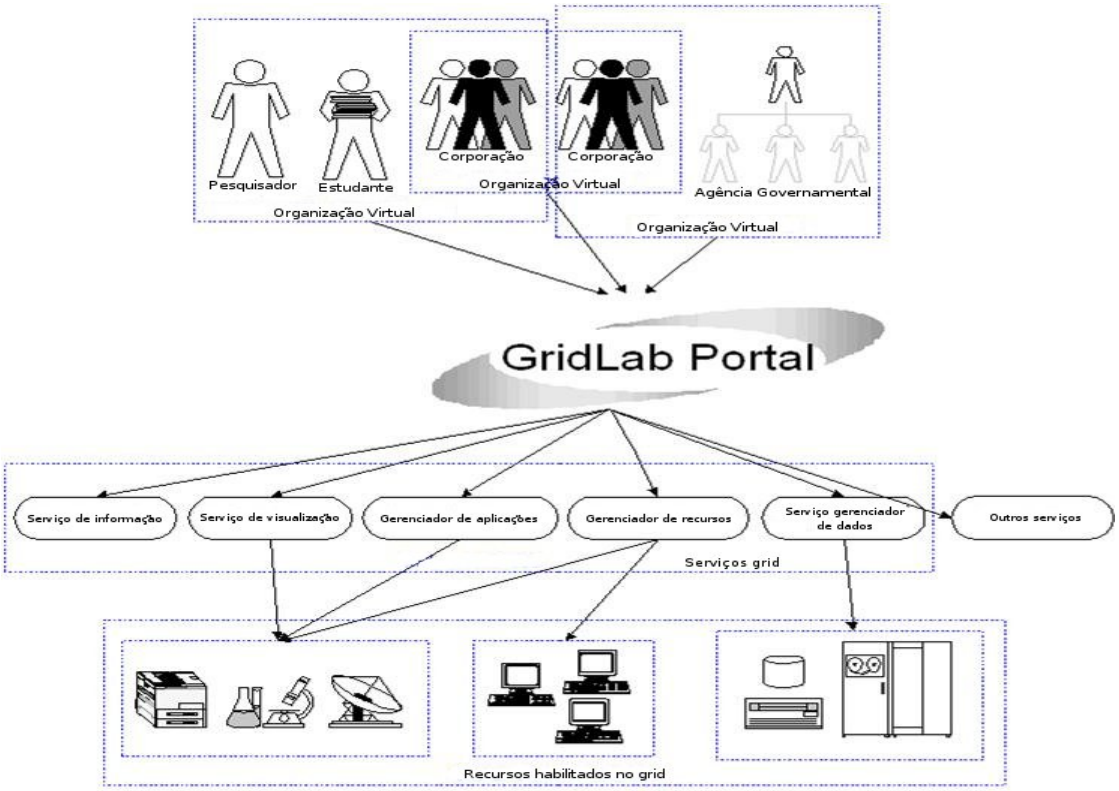

Figura 2.4: Outro exemplo de um *grid* [GRI 04]

define o que são serviços de *grid*, assim como os serviços e estruturas que um ambiente *grid* pode prover. O GGF ainda mantém a especificação da *Open Grid Services Infrastructure* (OGSI), que é a especificação formal da OGSA. A OGSI especifica um conjunto de serviços primitivos, definindo o núcleo de comportamento comum para todos os serviços grids [FER 03, JAC 03, Int 04, UNG 03, TUE 03]. A implementação padrão adotada pelo GGF para demonstrar a OGSA é o *Globus Toolkit*[Uni 04].

2.2.2 Tipos de *grid*

Jacobs [JAC 03] menciona que computação em *grid* pode ser usada de várias maneiras para atingir os requisitos dos programas. Os *grids* podem ser agrupados de acordo com o tipo de solução que eles melhor atingem. Cabe salientar que essa classificação não é rígida. Freqüentemente, os *grids* podem ser classificados em três tipos. Esses grupos primários podem ser os seguintes:

a) *Grid* computacional - está focado principalmente em agrupar recursos para aumentar o poder computacional. Neste tipo, a maioria das máquinas são servidores de alto desempenho. Exemplo: NASA *Information Power Grid* (IPG) [sit 04d] e *The Distributed ASCI Supercomputer* (DAS) [sit 04b];

b) *Scavenging grid* - o tipo mais comum adotado pelos usuários de máquinas do tipo estação de trabalho. Este tipo fica na busca por ciclos livres de CPU (Central Processing Unit) ou por outros tipos de recursos; e

c) *Grid* de dados - é responsável por guardar e prover acesso aos dados através de múltiplas instituições. Usuários não se preocupam onde os dados se encontram, desde que eles consigam acessá-los. Exemplo: *Particle Physics Data Grid* (PPDG) [Sci 04] e EU DataGrid [sit 04a].

2.2.3 Comparação entre *Cluster* e *Grid*

Uma definição de *cluster* foi redigida na seção 2.1.1. Dantas [PIT 03] apresenta um quadro que está reproduzido no quadro 2.3, com a omissão da coluna de

configuração oportunista e com a inclusão de uma característica (Acoplamento). A quantidade de nós em um *cluster* pode chegar na casa dos milhares, enquanto no *grid* pode chegar na casa dos milhões. Nota-se o compartilhamento do *grid* entre vários domínios, diferentemente do *cluster*, que fica restrito a apenas um. Esse compartilhamento no *grid* possibilita a divisão, entre eles, do custo, da segurança, do gerenciamento e ainda possibilita o uso de sistemas operacionais distintos. Quando se trata do acoplamento, no *cluster* é considerado forte, se comparado com o *grid*, pois o *cluster* está restrito ao tamanho de uma sala, enquanto os nós de um *grid* podem estar geograficamente distribuídos, enfraquecendo o acoplamento entre eles.

Característica	***Cluster***	***Grid***
Domínio	Único	Múltiplos
Nós	Milhares	Milhões
Segurança do processamento e recurso	Desnecessária	Necessária
Custo	Alto, pertencente a um único domínio	Alto, dividido entre domínios
Granularidade do problema	Grande	Muito Grande
Sistema operacional	Homogêneo	Heterogêneo
Gerenciamento	Centralizado	Descentralizado
Acoplamento	Forte	Fraco

Quadro 2.3: Comparação entre *Cluster* e *Grid*

Cabe destacar que uma das principais diferenças das características entre essas tecnologias é o fato de que o gerenciamento dos recursos envolvidos no *cluster*, geralmente, é de responsabilidade de uma instituição, e no caso do *grid* pode-se ter uma infra-estrutura gerenciada por diversas instituições. Cada uma se responsabiliza pelos seus recursos disponíveis no ambiente, utilizando normas, contratos ou acordos para a

interoperabilidade deles. Embora no *grid* possa ocorrer a divisão do custo entre as instituições, o custo é alto para as duas tecnologias porque é diretamente influenciado pelo seu ambiente heterogêneo, pela necessidade maior de segurança, pela infra-estrutura necessária para interligar recursos geograficamente distribuídos e também pela gerência dessa infra-estrutura.

2.3 Resumo

Este capítulo abordou, de forma resumida, a computação de alto desempenho. Esta vem sendo cada vez mais utilizada em diversos segmentos da sociedade. Entre as várias infra-estruturas existentes, foram abordados os computadores em *cluster* e a infra-estrutura de *grid*. A escolha por essas abordagens se deve ao objetivo desta dissertação, que fornece uma pequena fundamentação nessa área. No capítulo seguinte será tratada a simulação, a qual apresenta conceitos para o desenvolvimento da ferramenta proposta.

Capítulo 3

Simulação

Desde a década de 1960 pode-se encontrar livros que tratam sobre o tema simulação. Naylor [NAY 66] a conceitua da seguinte forma:

> A simulação é uma técnica numérica para conduzir experimentos em um computador digital, que envolve alguns tipos de modelos matemáticos e lógicos que descrevem o comportamento de um sistema de negócio ou econômico sobre um determinado período de tempo real.

Muitas outras definições já foram elaboradas. Entre elas se destacam a de Banks [BAN 99], que cita que "uma simulação é uma imitação de uma operação de um sistema ou processo do mundo real através do tempo", e a de Fujimoto [FUJ 00], que comenta que "uma simulação é um sistema que representa ou emula o comportamento de outro sistema através do tempo". Outros conceitos podem ser consultados em Freitas [dFF 01], Fishwick [FIS 95], Law [LAW 92] e Zeigler [ZEI 00].

Considerando a definição de Fujimoto, esta se apresenta de uma forma genérica, por isso pode ser aplicada às simulações que são feitas de forma manual ou às feitas por um computador e pode representar o comportamento de outro sistema igualmente genérico. Isto é, pode-se ter um programa de computador emulando o comportamento de outro computador ou de um outro sistema qualquer.

A simulação pode ser empregada em diversas áreas onde existe a necessidade de se realizar algum estudo sobre determinado modelo de sistema. Esses sistemas

podem ser econômicos, de manufatura, sociais, computacionais, entre outros. Essa presença em áreas distintas demonstra a natureza multidisciplinar do estudo da simulação. Ela pode ser usada para realizar alguns estudos sobre determinados sistemas e trabalhar com questões “E se...?”. Freitas [dFF 01] e Bratley [BRA 87] apresentam alguns fatores que levam os analistas a optar pelo uso da simulação. Esses fatores são apresentados nos itens abaixo.

- **Tratabilidade**: é muito caro ou leva muito tempo para mexer com o sistema real. O sistema a ser estudado pode não existir.
- **Treinamento**: simuladores de vôo para pilotos ou jogos de negócio para administradores.

3.1 Simulação discreta

A simulação discreta consiste em estudar modelos cuja representação se dá por meio de um conjunto de variáveis e cujas alterações de seus estados ocorrem em pontos específicos no tempo simulado. Nesses pontos específicos do tempo podem ocorrer diversos eventos, e são nos eventos que os estados das variáveis se alteram. Com isso, a simulação discreta pode ser também referenciada como simulação de eventos discretos. Elas podem ser realizadas manualmente, com o uso de papel e lápis, com o auxílio de calculadoras, ou podem ser executadas em computadores.

Zeigler [ZEI 00] apresenta uma especificação formal para aplicar a simulação de sistemas discretos. Essa especificação, denominada de *Discrete Event System Specification* (DEVS), apresenta uma estrutura básica mostrada em 3.1. Além do DEVS, Zeigler também apresenta o DESS (*Differential Equation Specified System*) e o DTSS (*Discrete Time Specified System*) como formalismos básicos para outros derivados desses. Teoremas, provas e detalhes adicionais sobre os formalismos podem ser obtidos em Zeigler [ZEI 00].

$$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta), \tag{3.1}$$

onde:

X é o conjunto de entradas,

Y é o conjunto de saídas,

S é o conjunto dos estados,

$\delta_{ext} : Q \times X \rightarrow S$ é a função de transição de saída,

$\delta_{int} : S \rightarrow S$ é a função de transição de entrada,

$\lambda : S \rightarrow Y$ é a função de saída,

$ta : S \rightarrow \Re_0^+ \cup \infty$ é a função de avanço do tempo,

$Q = \{(s, e) | s \in S, 0 = e = ta(s)\}$ é o conjunto total de estados.

3.2 Modelos

Os modelos são representações ricas em detalhes ou otimizadas de algum sistema real para representar seu comportamento. Os modelos podem ser construídos com vários níveis de abstração. O analista pode ser detalhista em certas partes do modelo e em outras usar um nível alto de abstração para representar o comportamento dessas partes do sistema. Esses sistemas podem ser de manufatura, computacional, de transporte, administrativos, entre outros. Freitas [dFF 01] apresenta uma classificação de modelos de acordo com seu propósito:

a) modelos voltados à previsão - usados para prever o estado do sistema em algum momento do futuro. Baseia-se no comportamento atual do sistema;

b) modelos voltados à investigação - usados para a busca de informações e para o desenvolvimento de hipóteses sobre o comportamento do sistema; e

c) modelos voltados à comparação - de acordo com os resultados das diversas replicações da simulação de um modelo, pode-se realizar comparações do efeito da troca de valores de certas variáveis, a fim de se observar o comportamento do sistema.

Kelton [KEL 02] apresenta a seguinte classificação:

a) Estático versus Dinâmico - no modelo estático o tempo não avança de forma natural, diferentemente do dinâmico. No estático o sistema é representado em um ponto particular do tempo. Já no modelo dinâmico representam-se as mudanças que ocorrem no sistema ao longo de um período;

b) Discreto versus Contínuo - no modelo contínuo, seu estado muda constantemente ao decorrer do tempo simulado. No sistema discreto, as mudanças de estado ocorrem em certos pontos ao longo do tempo simulado; e

c) Determinístico versus Estocástico - modelos determinísticos não possuem entradas randômicas, portanto suas saídas também não são randômicas. Os modelos estocásticos possuem entradas com valores randômicos gerando saídas também randômicas.

Cabe salientar que nem sempre um sistema discreto pode ser representado por um modelo discreto e isso também vale para sistemas contínuos. Optar pelo modelo vai depender do modo de estudo do analista sobre o sistema de interesse [BAN 99]. Nesta dissertação o foco está em modelos dinâmicos, discretos, determinísticos e/ou estocásticos.

Muitas vezes não basta para o analista apenas construir o modelo. Ele deve ter maneiras de validá-lo para que seu resultado esteja o mais perto possível da realidade. Para fazer essa validação ele pode proceder com alguns dos seguintes testes descritos por Bratley [BRA 87]: verificação manual da lógica, teste modular, verificação com soluções conhecidas e teste de *stress*. Na representação de modelos pode-se adotar pontos de vista (*world views*) que podem ser por processo, atividade ou eventos [FIS 95].

3.3 Modos de representar os modelos

Para criar modelos, o analista pode utilizar vários modos/tipos de modelagem para fazer a representação. Cabe ao analista escolher o mais apropriado. Fishwick [FIS 95] apresenta diversos modos de representar os modelos. Cada tipo de modelagem

se destaca de outros para representar certos aspectos de um sistema ou o sistema completo. Pode-se usá-los simultaneamente para melhor representar o sistema. Esses tipos de modelagem podem ser:

a) Modelagem Conceitual - é usada em modelos em que ainda não está claro o papel de cada componente nele envolvido. Seu nível de abstração é alto. Essa modelagem é, normalmente, a primeira a ser criada. Ela pode facilitar a comunicação entre o analista e a pessoa interessada na modelagem;

b) Modelagem Declarativa - é focada em estados e eventos. Sua representação se assemelha a um grafo. Deve-se visualizar o sistema como uma seqüência de mudanças de estados;

c) Modelagem Funcional - é usada em modelos que contêm como componentes principais as funções e as variáveis. Trabalha com a abordagem baseada na função cujo foco está nas funções de transição dos dados de entrada (incluindo as mudanças internas) para as variáveis de saída. Uma outra abordagem é a baseada na variável cujo foco está nas variáveis que são afetadas pelas funções;

d) Modelagem Espacial - usada em modelos que refletem uma decomposição do espaço. São úteis quando se divide o sistema em partes. Entendendo o comportamento das partes, compreende-se o sistema. Exemplos: modelos de plantas, dinâmica de gás, entre outros; e

e) Multimodelagem - composta do conjunto dos tipos apresentados, uma modelagem híbrida.

3.4 Tempo

O tempo em uma simulação pode ser expresso por uma variável inteira ou de ponto flutuante. Na simulação discreta o tempo evolui em intervalos irregulares/arbitrários, curtos ou longos. Na simulação contínua o tempo evolui em intervalos

regulares e prefixados. Estes podem ser regulados de acordo com a necessidade, mas sempre em intervalos constantes.

Há uma diferença quando se refere ao tempo de simulação e ao tempo real simulado. O tempo simulado é o tempo que transcorreu internamente na simulação; por exemplo, uma simulação executou um modelo que avaliava o funcionamento de um *call center* por duas horas. O tempo real simulado seria o tempo gasto para processar a simulação; por exemplo, a simulação de duas horas do *call center* foi executada em quinze segundos.

Para avançar o tempo da simulação, alguns mecanismos são usados. Conforme comentário anterior, em modelos contínuos o tempo avança de modo constante, isto é, em intervalos regulares, sendo eles curtos ou longos. Esse tipo de avanço está muito presente em simulações do clima, movimentos de veículos, fluxo do ar em volta de uma asa de avião, entre outros. O outro modo de avanço do tempo ocorre em modelos discretos em que o relógio avança de maneira irregular, com intervalos aleatórios, isto é, o relógio sempre avança para o tempo do evento que está agendado para ocorrer e este pode ser curto, em relação ao tempo atual, ou longo; exemplos: a simulação de um rota de vôo, uma rede de computadores, em que os eventos ocorrem em pontos bem definidos, como decolagem e pouso de um avião, início de embarque de passageiros, chegada de uma requisição no servidor, processamento de um pacote no roteador, entre outros.

3.5 Simulação discreta paralela e distribuída

A simulação discreta paralela e distribuída é uma tecnologia que possibilita a execução de modelos em sistemas computacionais paralelos/distribuídos [FUJ 00]. Uma simulação paralela é voltada a ser executada em computadores multiprocessados. A simulação distribuída está voltada a executar os modelos em computadores geograficamente distribuídos [FUJ 99]. Existem alguns fatores que influenciam a execução dos modelos em múltiplos computadores. Fujimoto [FUJ 00, FUJ 99] e Zeigler [ZEI 00] listam alguns deles:

a) redução no tempo de execução/aumento de velocidade;

b) aumento no tamanho do modelo;

c) interoperabilidade e compartilhamento de recursos;

d) integração de simuladores que são executados em computadores de diferentes fornecedores;

e) tolerância a falhas, etc.

Fujimoto [FUJ 00] menciona que, atualmente, duas classes de aplicações da simulação têm recebido mais atenção:

a) Simulações Analíticas - esse tipo de aplicação normalmente está preocupado em capturar dados detalhados sobre o sistema que será simulado. Ainda requer que o modelo reproduza, com a máxima exatidão, o comportamento do sistema a ser simulado para que as estatísticas geradas sejam válidas. É a forma clássica de simulação; e

b) Ambientes Virtuais - o objetivo central deste tipo de aplicação é o de dar a sensação aos usuários de se sentirem inseridos no sistema modelado. Quase sempre incluem a participação dos usuários como *entidades* dentro da simulação.

Dentro da simulação analítica, Jacobs [JAC 02] faz uma classificação quanto ao nível de distribuição:

a) ambientes de simulação integrados - esse ambientes não são desejáveis para o uso distribuído, uma vez que o sistema de visualização está totalmente atrelado ao ambiente de execução da simulação. Principais características: único usuário, computador e *thread*;

b) ambientes de simulação semidistribuídos - compartilham características baseadas na Internet. Podem ser divididos em duas subclasses: (1) a simulação é executada no cliente; ou (2) a simulação é executada no servidor e os dados de saída são recebidos pelos clientes; e

c) ambientes de simulação totalmente distribuídos - consistem na distribuição da simulação e dos serviços de visualização. Esses ambientes provêem integração com sistemas de informações distribuídas e inúmeras variações na configuração.

Cabe salientar que nem tudo é perfeito. Problemas podem ser encontrados no momento da execução tanto no modelo quanto na configuração adotada para realizar a execução. Murphy [MUR 02] comenta alguns sucessos e falhas no desenvolvimento de simulações. Bagrodia [BAG 96] comenta algumas ciladas que devem ser evitadas pelo analista para conseguir projetar uma implementação paralela eficiente:

a) cuidados com variáveis compartilhadas;

b) atenção com ponteiros para estruturas de dados;

c) ciclos com *delay* zero;

d) carga sem equilíbrio nos nós;

e) alto tráfego de mensagens; e

f) baixo paralelismo.

Fishwick [FIS 95] menciona que, em geral, todos os tipos de modelos são apropriados para execução em paralelo. Mas, na prática, os melhores modelos para serem executados em paralelo são aqueles com modelagem espacial e os com modelagem funcional (seção 3.3). Para executar os modelos com os programas de simulação discreta nos ambientes paralelos ou distribuídos, existem duas abordagens de protocolos de sincronização: quanto ao tipo de particionamento do modelo e quanto aos objetivos pretendidos na execução. A primeira abordagem, denominada *Single Replication In Parallel* (SRIP), possibilita a execução de uma única replicação de um modelo em paralelo. Já a segunda abordagem, denominada *Multiple Replication In Parallel* (MRIP), possibilita a execução de várias replicações em paralelo. Para executar o modelo usando a abordagem SRIP (figura 3.1), este é particionado em processos lógicos (PL), os quais são mapeados como processos físicos. Os processos lógicos se comunicam por meio de mensagens contendo

o tempo e o evento deste determinado tempo [FUJ 00, ZEI 00, NS 02]. Já no sistema MRIP (figura 3.2) todo o modelo está em um processo lógico, que está relacionado a um processo físico e que troca informações sobre as estatísticas obtidas com um controlador mestre.

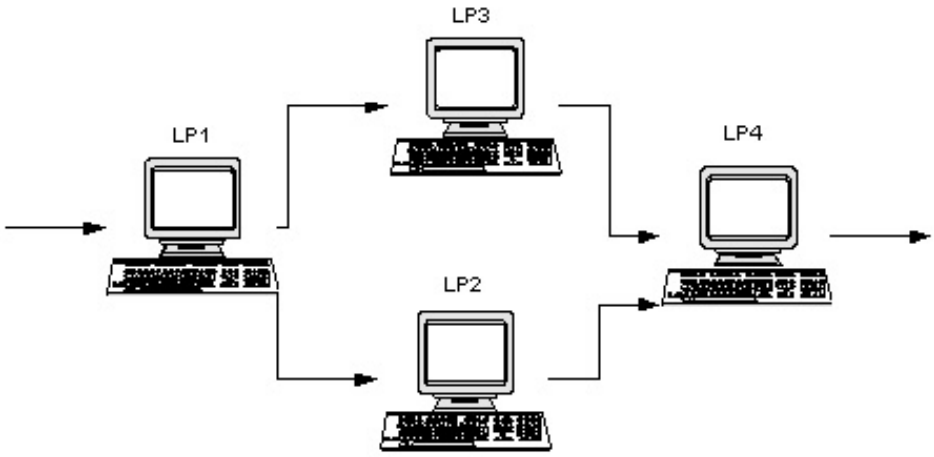

Figura 3.1: Simulação em modo SRIP

Existem protocolos de sincronização utilizados na execução do programa que trabalham para tentar prevenir que problemas no processamento de eventos não ocorram. Esses protocolos serão vistos com mais detalhes na seção a seguir.

3.5.1 Protocolos de sincronização

Os protocolos de sincronização podem ser classificados como síncronos ou assíncronos. Os assíncronos são os que mais recebem atenção da comunidade. Fishwick [FIS 95] cita dois tipos de métodos síncronos: compartilhamento do relógio (a cada atualização do relógio, todo o sistema é atualizado para o novo valor do relógio); e compartilhamento da lista de eventos futuros (onde se têm todos os objetos trabalham com uma única lista de eventos). Fujimoto [FUJ 00] cita que em alguns algoritmos assíncronos conservadores há como se trabalhar de modo síncrono mediante o uso de pontos

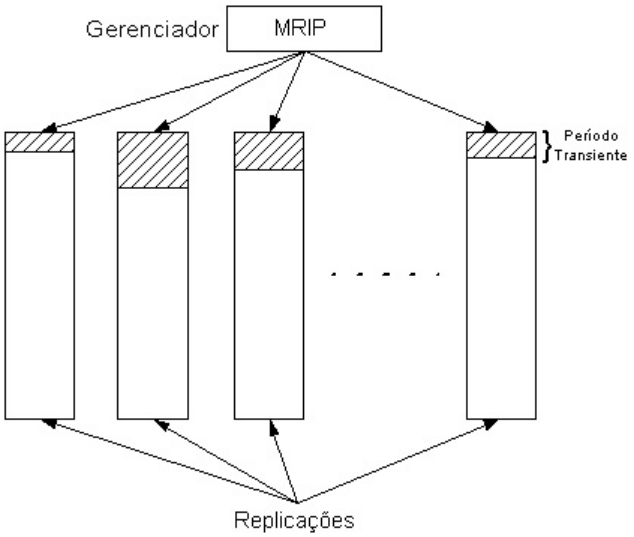

Figura 3.2: Simulação em modo MRIP [BRU 03]

de controle. Nos modelos assíncronos existe ainda uma subdivisão dos protocolos em conservadores e otimistas. Para esses tipos de protocolos existem algumas características desejáveis. Reynolds [Rey 88] define nove variáveis a que se deve dar atenção no momento de projetar um protocolo, que são:

a) particionamento - agrupar PLs com base em conjuntos de variáveis do projeto;

b) adaptabilidade - mudar as variáveis do projeto baseando-se no conhecimento de determinados aspectos do estado da simulação;

c) agressividade - processar mensagens com base em conhecimento condicional;

d) acuracidade - processar eventos em um PL na ordem correta;

e) risco - passar mensagens que tenham sido processadas com base na agressividade ou na imprecisão assumida no PL;

f) conhecimento embutido - incluir na simulação conhecimento sobre o comportamento dos atributos do PL;

g) disseminação de conhecimento - disseminar conhecimentos para outros PLs;

h) aquisição de conhecimento - requisitar conhecimento de outros PLs; e

i) sincronia - observar o grau de diferença de tempo entre os PLs.

O principal problema que pode ocorrer nas simulações analíticas é o de causa e efeito [BRU 03]. Esse problema, abordado por Fujimoto [FUJ 00], Fishwick [FIS 95] e Zeigler [ZEI 00], ocorre quando não é respeitada a ordem de processamento dos eventos.

3.5.1.1 Protocolos conservadores

Protocolos conservadores, ou conservativos, determinam quando é seguro que um evento seja processado, isto é, quando o evento a ser processado tem todos os eventos dele dependentes já processados [FUJ 99, NS 02]. Estes realizam esse processo para prevenir o problema de causa e efeito. Dois exemplos destes protocolos são:

a) CMB (Chandy, Misra e Bryant) - cada processo executa somente os eventos se tiver certeza de que mais nenhum evento chegará com o tempo menor do que vai ser executado [NS 02]. Podem ocorrer *deadlocks* quando os processos tentam ter a certeza para executar o evento determinado. Para prevenir esses *deadlocks* utilizam-se mensagens nulas com o tempo que é seguro para ser executado. Neste caso, como um PL está aguardando as mensagens para poder executar o evento com segurança, ele recebe uma mensagem apenas com o tempo. Esse processo é denominado *lookahead* [FUJ 00]; e

b) Lookback - proposto por Chen [CHE 02], é um algoritmo que tem a capacidade de fazer o PL voltar a um passado local.

3.5.1.2 Protocolos otimistas

Os protocolos otimistas processam os eventos na medida do possível, podendo ocorrer o problema de causa e efeito. Ao processar um evento, esse protocolo é otimista em relação à segurança de se processar determinado evento. No entanto, esse

protocolo provê um mecanismo de recuperação, para um estado seguro, caso ocorra um problema do tipo causa e efeito [FUJ 00, NS 02].

O protocolo que implementa esse tipo de abordagem é chamado de *Time Warp*. Esse protocolo tem a capacidade de calcular os tempos locais do PL, assumindo que as mensagens serão entregues sempre na ordem correta. Caso isso não ocorra, isto é, se chegar uma mensagem com o tempo anterior ao tempo local, então o PL executa um *rollback* para o evento anterior ao tempo que chegou a mensagem que ocasionou o *rollback*. Após voltar o tempo, o PL manda *antimensagens* para cancelar as mensagens inválidas por ele enviadas anteriormente à chegada da mensagem recebida [FUJ 00, ZEI 00, NS 02]. Variações deste protocolo podem ser vistas em Carothers [CAR 02].

3.5.2 Aplicações

Fujimoto [FUJ 99, FUJ 00] apresenta algumas aplicações do uso da simulação paralela distribuída:

a) militar: simulações de guerras, ambientes de treinamento e para testes e avaliações;

b) entretenimento: simuladores de vôo, jogos multiusuário;

c) área social: simulações ambientais, de parques, entre outros;

d) redes de telecomunicações;

e) sistemas computacionais e circuitos lógicos digitais.

3.6 Trabalhos relacionados

A seguir são apresentados alguns trabalhos que de alguma maneira estão relacionados o projeto desenvolvido nesta dissertação.

- Em Jacobs [JAC 02] foi desenvolvida uma biblioteca, denominada *Distributed Simulation Object Library* (D-SOL), para executar simulação distribuída. Esta biblioteca utiliza a linguagem Java.

- Em Rocha [dR 02] é apresentada uma forma de desenvolver simuladores através de componentes. O desenvolvimento foi realizado utilizando-se a linguagem Java.

- Em Bruschi [BRU 00, BRU 03] é proposto um ambiente de simulação distribuída automático (ASDA). Neste ambiente são utilizadas várias tecnologias para o desenvolvimento.

- Em Carothers [CAR 99] é proposto um modo para realizar visualizações de simulações paralelas.

- Em Vadhiyar [VAD 03] foi desenvolvido um *framework* para uma migração eficiente para o *grid*.

- Em Arief [ARI 00] é apresentado um gerador de programas de simulação que utiliza as definições da UML para interagir com o usuário. O gerador foi desenvolvido utilizando-se as tecnologias Java e XML.

- Em Bumble [BUM 02] é apresentada uma arquitetura para simuladores distribuídos não-determinísticos.

- E em *Grid-Enabled Medical Simulation Services* (GEMSS) *Project* [sit 04c] são apresentados alguns serviços de simulações médicas que são acessadas/executadas em *grid*.

3.7 Resumo

Alguns conceitos relativos à simulação foram abordados neste capítulo. Tópicos que envolvem os fundamentos da simulação foram brevemente discutidos. Além destes tópicos fundamentais, conceitos sobre a simulação discreta paralela e distribuída

foram delineados. O próximo capítulo trata da especificação da ferramenta e do desenvolvimento de uma opção de implementação dela para esta dissertação.

Capítulo 4

Editor visual integrado ao ambiente *grid*

Este capítulo aborda, inicialmente, uma especificação de uma ferramenta para o desenvolvimento e execução de modelos discretos usando um ambiente de alto desempenho, como exemplo, *grid*. A seguir aborda-se a ferramenta desenvolvida nesta dissertação baseando-se na especificação apresentada. Essa ferramenta foi denominada GridSimulator. Para uma melhor integração com o ambiente *grid*, ela foi dividida em dois projetos principais: o editor visual de modelos; e o programa que executa os modelos. Primeiramente, será abordado um aspecto geral da ferramenta e, depois, os aspectos referentes ao editor e ao processador de modelos.

Visando a fornecer maior flexibilidade para o usuário, o projeto foi dividido em editor e processador. O primeiro trata de prover um ambiente para criação de modelos, e o segundo trata de executar o modelo criado e gerar os dados de saída. Com isso, o usuário tem mobilidade para desenvolver seu modelo no lugar onde seja melhor para ele trabalhar e, então, com o modelo pronto, pelo uso da *internet* ou de alguma rede de onde se tenha acesso ao *grid*, ele o envia para ser processado no ambiente *grid*. Um esquema de exemplo está apresentado na figura 4.1. A comunicação entre o editor e o processador se dá pelo emprego da *Extensible Markup Language* (XML). O editor gera um arquivo no formato XML e este é transferido para o nó do *grid*, onde será executado

pelo processador, que faz a leitura do modelo e o processa. A ferramenta foi desenvolvida na língua inglesa, pois se pretende disponibilizá-la na comunidade de software livre mundial.

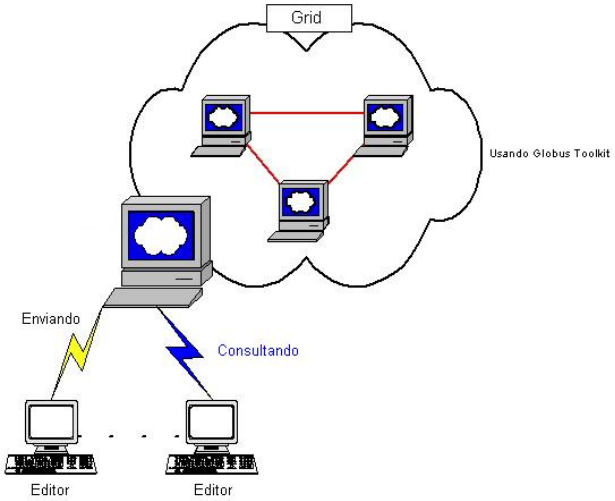

Figura 4.1: Esquema do GridSimulator

4.1 Especificação de uma ferramenta para construção de modelos discretos

Uma ferramenta para desenvolver modelos discretos usando ambientes de alto desempenho necessita simplificar a execução deles para seu usuário final. O uso desses ambientes no desenvolvimento de modelos apresenta algumas características, entre as quais o computador do usuário não faz parte do ambiente de alto desempenho; o usuário não está constantemente conectado ao ambiente; e o ambiente de alto desempenho possui mais poder computacional que o computador utilizado pelo usuário. Com o objetivo de atender a essas características foi criada uma especificação que apresenta uma divisão entre a parte responsável por modelar e a parte responsável por executar o modelo.

Um diagrama de exemplo foi montado para exemplificar a interação

entre os possíveis módulos existentes em uma ferramenta de modelagem. Esse diagrama (figura 4.2) apresenta as áreas de dados: modelo e animação, apresenta também os seguintes módulos: interface gráfica, otimização, modelagem, animação e execução. Mais módulos, com diferentes objetivos, podem surgir durante a evolução desta especificação. No entanto, cada módulo do diagrama possui uma tarefa específica:

a) **Interface gráfica**: a tarefa deste módulo é a de fornecer um meio de iteração com o usuário para facilitar o uso da ferramenta e a de fornecer acessos gráficos aos outros módulos que estão agrupados a este;

b) **Otimização**: este módulo está focado na tentativa de otimizar os modelos de acordo com as necessidades do usuário; por exemplo, para uma execução em menor tempo, para a diminuição de componentes de modelagem ou para diminuir o número de mensagens entre os processos lógicos (caso estiver simulando paralelamente com SRIP);

c) **Modelagem**: este deve prover maneiras de melhor criar os modelos discretos dos usuários, utilizando o ponto de vista por processo. Deve fornecer os componentes de modelagem, maneiras de gerenciá-los, alinhá-los, ocultá-los ou não, entre outras tarefas comumentes encontradas; por exemplo, copiar e colar de componentes, edição de propriedades dos componentes, validação do modelo, etc;

d) **Animação**: este módulo é responsável em prover a animação do modelo criado. A animação poderia ocorrer paralela à simulação se esta fosse feita no computador do usuário, isto é, local e seqüencialmente. No entanto, caso a simulação ocorresse remotamente, a animação poderia ser realizada após o término da simulação com base em dados gerados pelo simulador para esse fim; e

e) **Execução**: a tarefa deste módulo seria prover os meios de execução dos modelos discretos criados. Os meios de execução poderiam ser seqüencial, paralelo ou distribuído. O usuário escolhe qual meio iria utilizar para a execução. Este módulo teria suporte para gerar dados que seriam utilizados para a realização da animação do modelo executado. Este módulo se encontra tanto no computador cliente como no computador/ambiente remoto, pois a possibilidade da execução local não é descartada. A

execução local pode ser utilizada pelo usuário como forma de testar o modelo antes de enviá-lo para a execução no computador remoto.

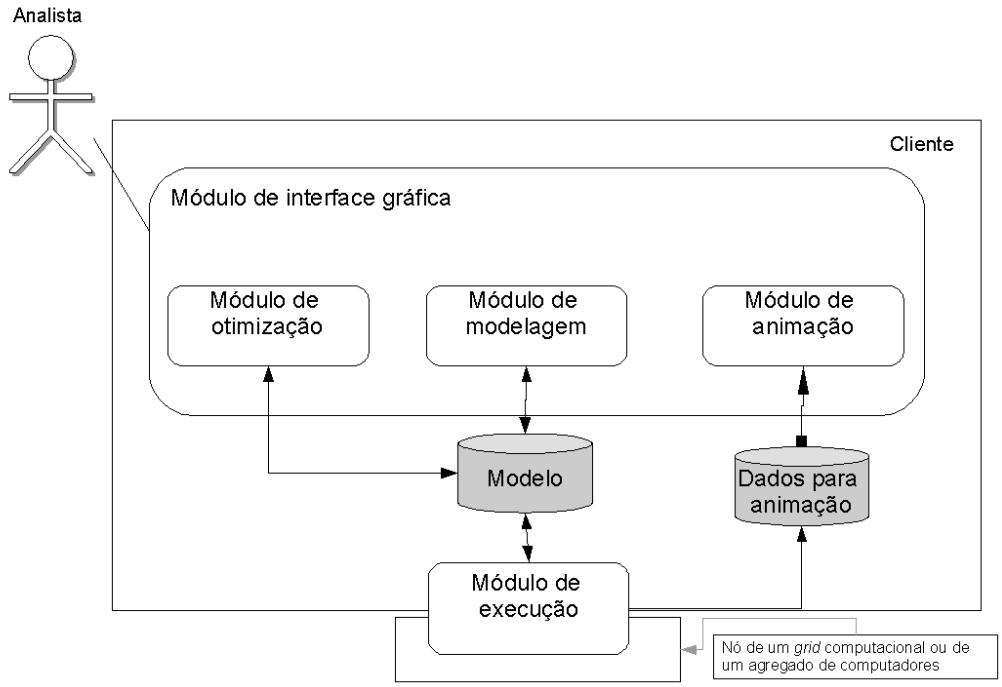

Figura 4.2: Interação geral

Do ponto de vista do analista/usuário a ferramenta deverá suportar: o gerenciamento do modelo; o envio do modelo para execução e acompanhamento de seu processamento quando executado remotamente. Para descrever essa relação do analista com a ferramenta têm-se os diagramas simplificados de caso de uso para modelagem (figura 4.3) e para execução e gerenciamento do modelo (figura 4.4). No primeiro diagrama, figura 4.3, o analista realiza o gerenciamento do modelo. Este inclui os casos abrir, criar e manipular modelos pela ferramenta. No diagrama seguinte, figura 4.4, o usuário pode executar o modelo seqüencialmente, pode enviar para execução remota e pode gerenciar os modelos enviados para execução remota. A execução remota pode ser em agregados de computadores e em grades computacionais.

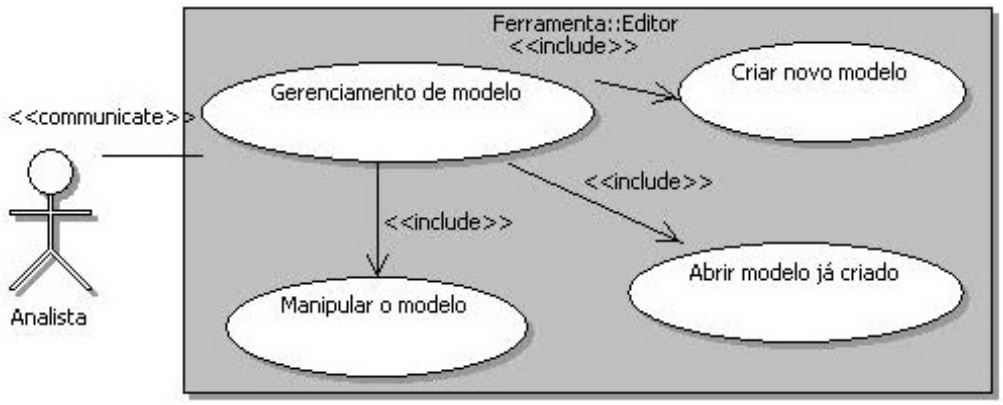

Figura 4.3: Diagrama de caso de uso para modelagem

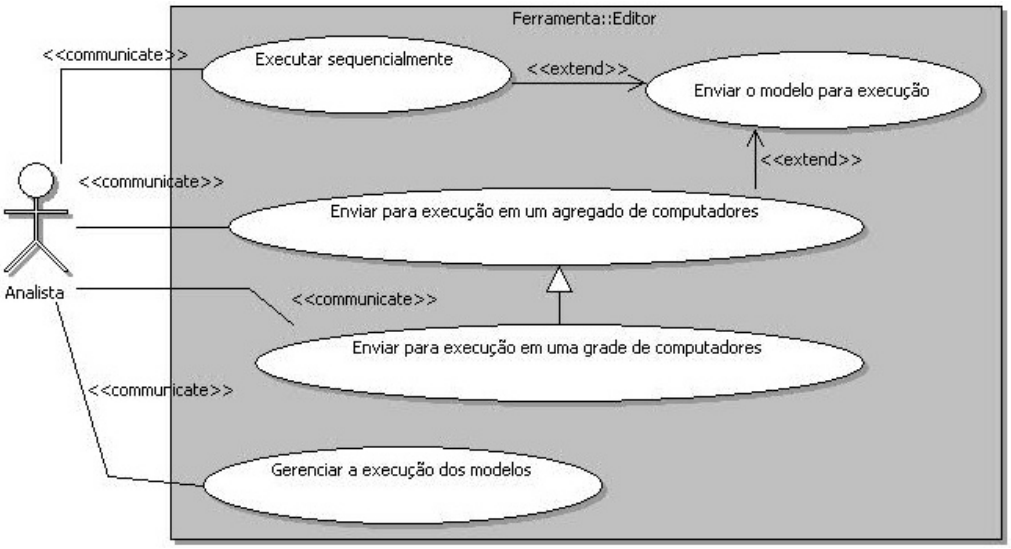

Figura 4.4: Diagrama de caso de uso para envio

Para exemplificar a configuração de uma ferramenta desse tipo, apresenta-se o seguinte diagrama de implantação (figura 4.5). O diagrama apresenta as relações entre o nó cliente e o nó remoto, através do uso de conexão TCP/IP. No nó cliente há o componente Editor, que cria os modelos, e no nó remoto está o componente de execução (Processador). O Editor cria o modelo e o transfere para o nó remoto através da interface de escrita. O componente Processador lê o modelo, o processa e gera o relatório final, o qual é utilizado pelo Editor para apresentar o resultado final da simulação. Esta configuração não é predominante, haja vista que se pode ter o componente de execução no nó cliente também.

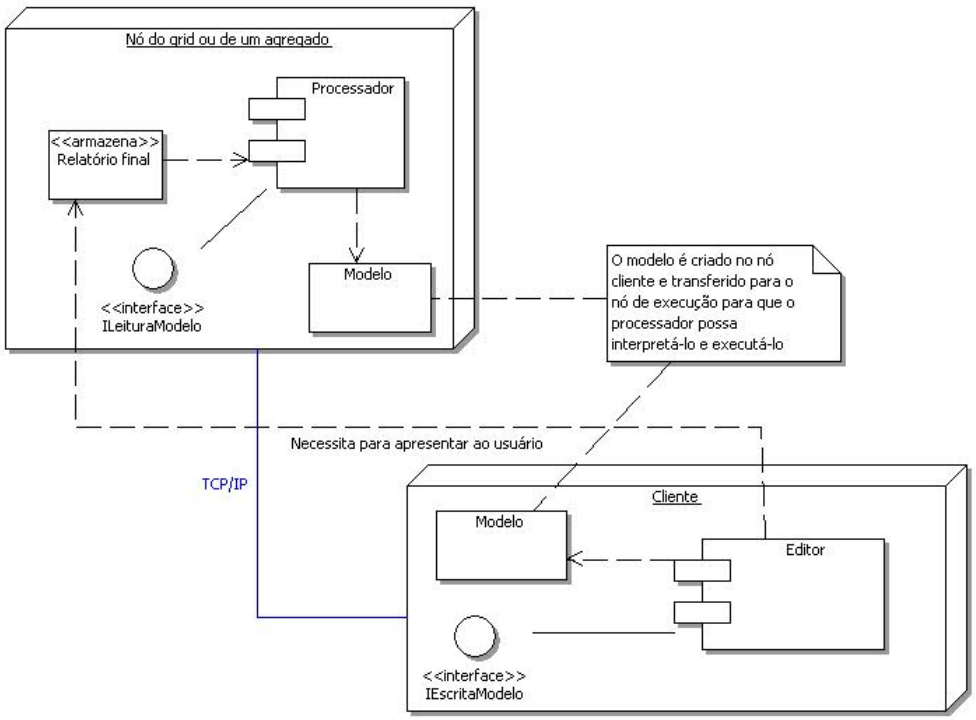

Figura 4.5: Diagrama de implantação de uma ferramenta de modelagem

4.1.1 Detalhes do Editor e do Processador

Com o objetivo de detalhar o Editor apresenta-se um diagrama de classes (figura 4.6). Neste diagrama encontra-se as principais classes do editor e suas respectivas relações. São classes-chave ou principais do diagrama a classe InterfaceGrafica, Modelo, ComponenteModelagemBase, GerenciadorModeloRemoto, EnvioRemoto, EnvioSequencial, ESModelo, Comentario e Conector. Cada classe-chave tem a seguinte função:

a) **InterfaceGrafica**: esta classe representa a interface gráfica. Ela fará a iteração com o usuário para realizar as ações disponíveis a ele;

b) **Modelo**: esta classe representa o modelo discreto que o usuário poderá construir e manipular. Ela agrega ainda comentários, conectores de componentes e os componentes. Esta disponibiliza método para validar o modelo e atributos referentes à execução; por exemplo, número de replicações, tempo para simular e a unidade desse tempo;

c) **ComponenteModelagemBase**: esta classe é uma interface mínima para se construir componentes que ficarão disponíveis para o usuário. Esta especifica que os componentes de modelagem devem ter métodos para validar, para realizar o desenho que representará o componente, para verificar se determinado componente pode se conectar e para gerenciar as conexões deste. Como componentes básicos, este diagrama prevê pelo menos o componente Requisita, Libera, Recurso, Servidor, Gerador, Entidade, Saida, Decisao, Atribuicao e o Retardo;

d) **GerenciadorModeloRemoto**: nesta classe deve-se possibilitar o gerenciamento dos modelos que foram enviados para execução remota. Entende-se por gerenciamento a possibilidade de verificar se a simulação alcançou seu fim, de poder copiar o relatório final para o computador do usuário para a visualização posterior e de remover arquivos, referentes a determinada simulação, do computador remoto;

e) **EnvioRemoto**: esta possibilita o envio do modelo para ser executado remotamente. Esta classe é uma interface para realizar esse envio. Suas implementações dependem

do programa que está sendo utilizado nos computadores remotos para gerenciar determinado ambiente de alto desempenho;

f) **EnvioSequencial**: esta classe é responsável por realizar a execução do modelo no computador local do usuário;

g) **ESModelo**: esta é responsável pelo armazenamento e recuperação do modelo. Ela converte o modelo para algum formato de arquivo, o qual, posteriormente, poderá ser salvo no computador local e realizar também a recuperação do modelo no formato salvo em arquivo para o formato que pode ser manipulado pelo usuário. Esta classe depende diretamente de como for implementada a classe Modelo;

h) **Comentario**: a classe comentário representa comentários que o usuário poderá inserir no modelo; e

i) **Conector**: esta classe representa a conexão entre os componentes do modelo.

Para possibilitar a execução do modelo tanto remotamente quanto no computador do usuário e de modo que o processamento ocorra seqüencial ou em paralelo, projetou-se o processador de modelos conforme apresentado no diagrama de classe da figura 4.7. A execução ocorre mediante o uso de uma lista de eventos futuros que ocorrerá no processo lógico corrente. Esta lista possui em cada entrada um evento que tem o tempo correto para executá-lo e o componente que executará o evento. Para detalhar as classes que compõem o processador, segue uma lista com detalhes de cada uma:

a) **Processador**: é o responsável por executar as diversas replicações do modelo. Ele também realiza a alteração das sementes da geração dos números aleatórios de cada processo lógico. Este também aciona o conversor;

b) **ProcessoLogico**: este representa um processo lógico. Nesta classe localiza-se o algoritmo de simulação escolhido. Neste caso, sugere-se o uso da simulação através de eventos. Este também é o responsável por finalizar a simulação no momento correto e gerar o relatório final;

c) **Modelo**: representa o modelo a ser executado. Este é montado pelo conversor. É responsável pela geração dos eventos iniciais de cada componente. O modelo contém uma lista dos componentes e outra das variáveis dele;

d) **ComponenteModelagemBase**: é a interface para os componentes que serão criados para montar o modelo. Cada componente disponível na interface gráfica tem um correspondente no processador. Cada componente deve ter pelo menos o método para definir o comportamento de processamento, para geração de eventos iniciais e um acesso ao seu sucessor, se houver. Os componentes básicos previstos neste diagrama são: Requisita, Libera, Recurso, Servidor, Gerador, Entidade, Saida, Decisao, Atribuicao e Retardo;

e) **Conversor**: esta classe é responsável em converter o modelo armazenado em arquivo para o modelo a ser processado pelo processo lógico;

f) **ListaEventoFuturo**: esta representa a lista de eventos futuros a ser processada pelo processo lógico para realizar a simulação; e

g) **Evento**: representa o evento a ser processado pelo componente nele indicado.

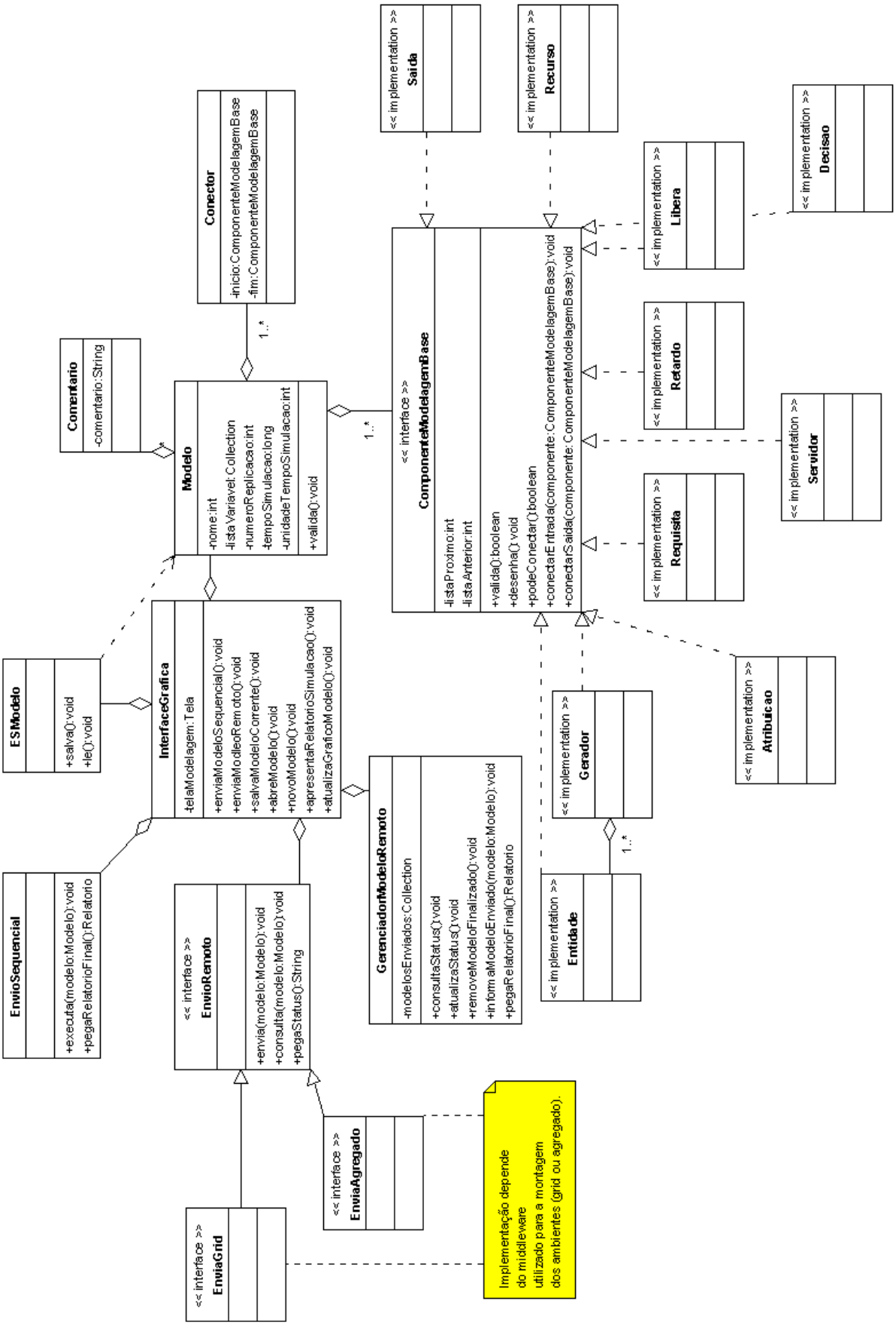

Figura 4.6: Diagrama de classe de um editor de modelos discretos

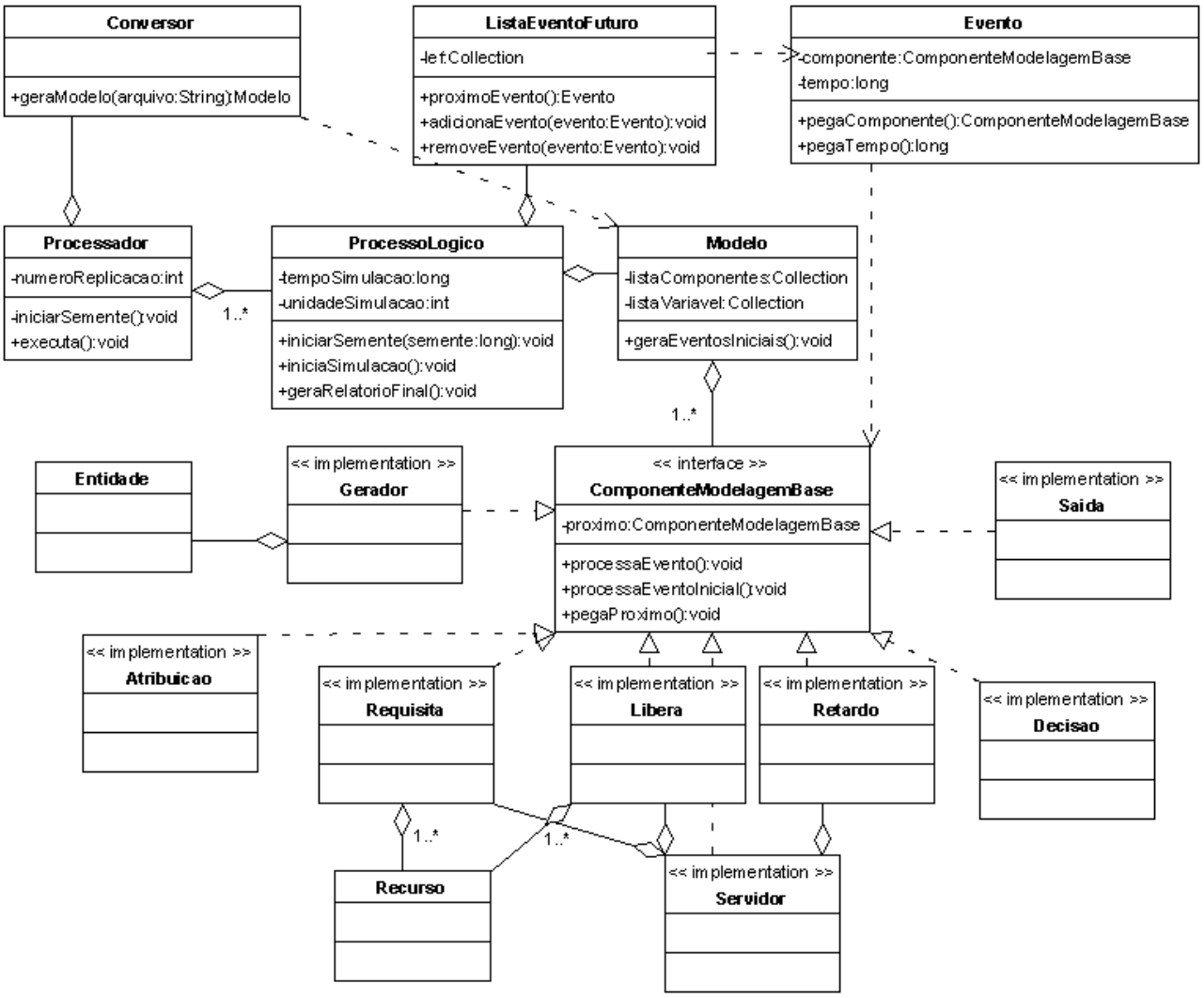

Figura 4.7: Diagrama de classe de um processador de modelos discreto

4.1.2 Detalhe comportamental dos componentes propostos

Para detalhar o comportamento dos componentes propostos na seção anterior foram desenvolvidos alguns diagramas para descrevê-los. Os componentes Entidade e Servidor não foram detalhados porque a Entidade não tem estado nem atividade. É ela que é manipulada pelos outros componentes, e o Servidor não é detalhado por ser uma agregação do Retardo, Requisita e Libera.

4.1.2.1 Atribuição

O componente de atribuição realiza a alteração do valor de determinados atributos pertencentes à entidade que está sendo manipulada por ele. Conforme expressado na figura 4.8, este recebe o evento de chegada da entidade, realiza os cálculos para a atribuição e faz as alterações dos valores de determinados atributos da entidade. Por fim, encaminha a entidade para seu sucessor.

Figura 4.8: Diagrama de atividades do processo de atribuição

4.1.2.2 Decisão

Este é o único componente especificado que pode realizar alteração no fluxo lógico que a entidade pode seguir no modelo. Ao receber a entidade, ele realiza um teste para decidir se encaminha a atual entidade para o sucessor relativo ao resultado positivo (Sim) ou se encaminha ao sucessor relativo ao resultado negativo (Não). Este processo é apresentado na figura 4.9.

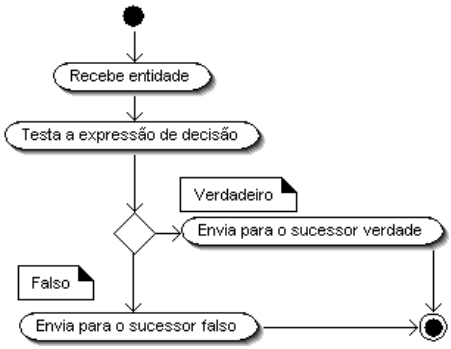

Figura 4.9: Diagrama de atividade do processo de decisão

4.1.2.3 Gerador

Este é responsável pela geração das entidades do modelo. Ele se comporta conforme a figura 4.10. Pode gerar mais de uma entidade por evento e, após realizar a geração, ele encaminha as entidades para seu sucessor no modelo.

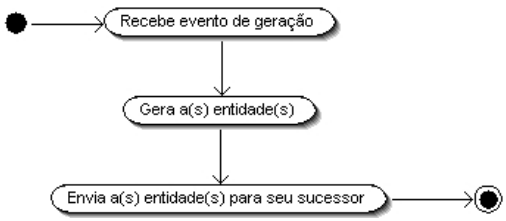

Figura 4.10: Diagrama de atividade do processo de geração

4.1.2.4 Libera

O Libera é responsável por liberar os recursos que foram requisitados pela entidade. Ele deve funcionar conforme o diagrama da figura 4.11. Ao receber o evento de chegada da entidade, libera os respectivos recursos e realiza a notificação para os componentes que estão aguardando pelos recursos liberados. Após executar esses passos, ele encaminha a entidade para seu sucessor no modelo.

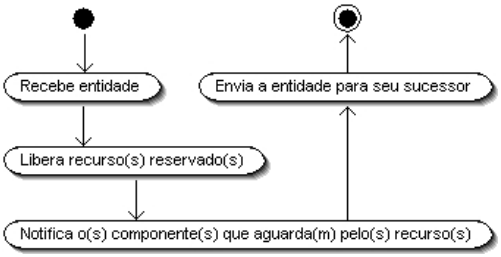

Figura 4.11: Diagrama de atividade do processo de liberação de recurso(s)

4.1.2.5 Recurso

O Recurso não tem uma seqüência de atividades, mas sim diferentes estados possíveis. As trocas entre estados estão especificadas no diagrama de estado na figura 4.12. Ele pode estar no estado:

a) **Livre**: quando está disponível para ser reservado a qualquer entidade;

b) **Ocupado**: quando está reservado a uma determinada entidade; e

c) **Falhado**: quando ocorreu uma falha e não pode ser reservado a nenhuma entidade. Se por acaso houver uma entidade ocupando o recurso no momento da falha, ela aguarda a volta da falha e continua o uso do recurso.

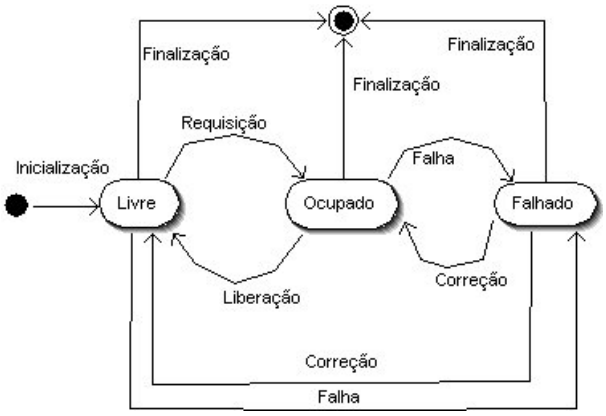

Figura 4.12: Diagrama de estado do recurso

4.1.2.6 Requisita

Realiza a requisição dos recursos necessários para que a entidade possa continuar seu fluxo no modelo. Este processo ocorre conforme o especificado na figura 4.13. Ele recebe a entidade e verifica a disponibilidade dos recursos. Caso tenha recursos disponíveis, eles são reservados e a entidade é encaminhada para o componente sucessor no modelo. Mas caso não tenha recursos disponíveis, a entidade é colocada na fila para aguardar a disponibilidade deles.

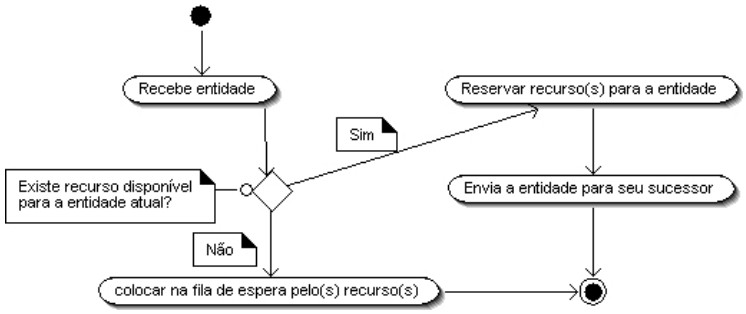

Figura 4.13: Diagrama de atividade do processo de requisição de(s) recurso(s)

4.1.2.7 Retardo

Neste componente ocorre um retardo no tempo da entidade, que é tratado conforme o diagrama da figura 4.14. O componente recebe o evento de chegada da entidade, faz o cálculo do tempo de retardo (Tr) e a encaminha para o sucessor deste componente no modelo, considerando o tempo de retardo em relação ao tempo corrente (Tc). Então, o tempo do evento (Te) de chegada no sucessor será $Te = Tc + Tr$.

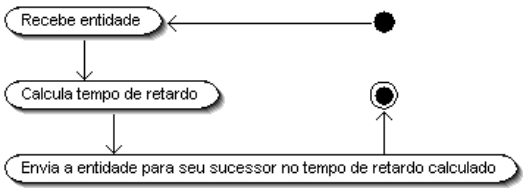

Figura 4.14: Diagrama de atividade do processo de retardo

4.1.2.8 Saída

Neste componente (figura 4.15) a entidade é retirada da execução do modelo. Ele recebe o evento de chegada da entidade, registra os dados de interesse e remove a entidade do modelo.

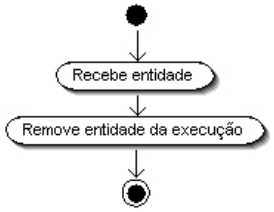

Figura 4.15: Diagrama de atividade do processo de saída

4.2 Editor visual

O GridSimulator Editor foi desenvolvido com o objetivo de prover ao analista uma ferramenta para construção de modelos para simulação discreta e este ainda provê uma integração com um ambiente de *grid* para a execução do modelo. A forma de construção dos modelos é realizada utilizando-se a "*world-view*" por processo (ver seção 3.2). Para o desenvolvimento do editor foi utilizada a plataforma Java [Sun 03]. Sua escolha se deve ao fato de seu código binário (*bytecode*) ser multiplataforma e pela quantidade de bibliotecas disponíveis para se trabalhar com interfaces gráficas. A figura 4.16 apresenta o Editor sem nenhum modelo aberto, pronto para iniciar a modelagem ou abrir um modelo criado previamente.

Conforme mencionado anteriormente, o Editor salva o modelo no formato XML. Para realizar a conversão do modelo gráfico para XML, foi utilizada a biblioteca para Java denominada JDOM [HUN 04]. Já em relação à interface gráfica, foi utilizado o *framework* chamado JHotDraw [GAM 04]. Essa biblioteca provê um conjunto de classes que possibilita maior velocidade no desenvolvimento de programas que utilizam gráficos/desenhos técnicos e estruturados. Um exemplo de aplicação desenvolvida utilizando este *framework* é o Analisador de Redes de Petri em Java (JARP). O projeto

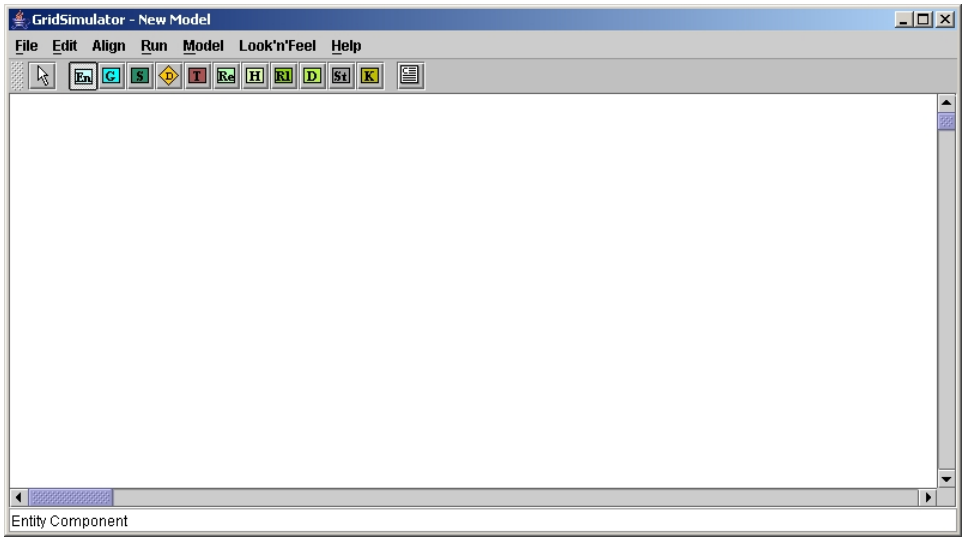

Figura 4.16: GridSimulator Editor

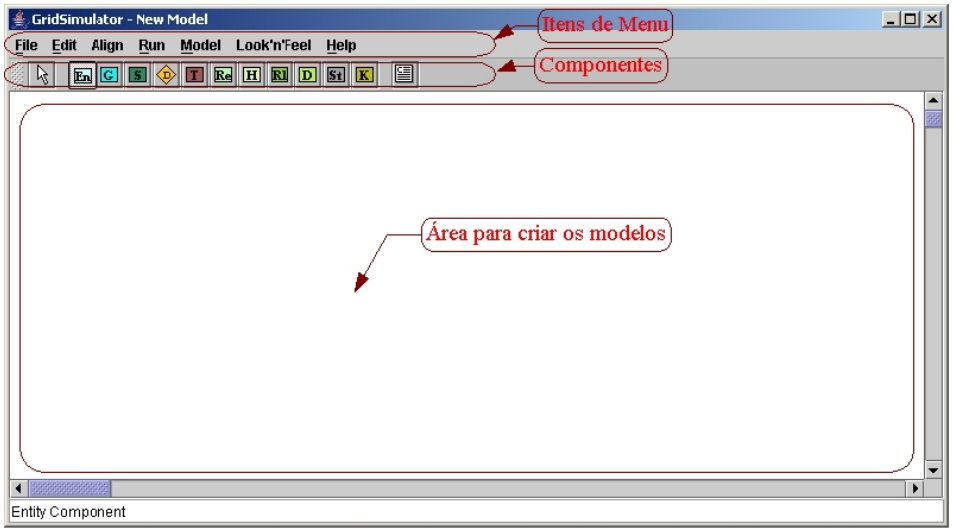

Figura 4.17: Áreas de uso do GridSimulator Editor

JARP foi desenvolvido pelo LCMI/DAS/UFSC para criar uma ferramenta de auxílio na análise de redes de Petri ARP [LCM 04].

A ferramenta tem três áreas de interesse para o usuário (figura 4.17): os itens de *menu*; a barra com os componentes; e a área de modelagem. Nos itens de menu há diversas opções para o analista. Por exemplo, pode-se alinhar de diversas maneiras os componentes, alterar algumas configurações do modelo, enviar para execução no *grid*, esconder/mostrar componentes (Entidade e Recurso) que não fazem parte da representação do modelo por processo, entre outras opções. Na barra de componentes o analista tem a opção de escolher os devidos componentes para a construção do modelo. Eles podem ser do tipo: *generator, server, decision, trash, resource, release, hold, delay, set, keep e comment*. A área de modelagem é o espaço que o analista tem para construir seu modelo.

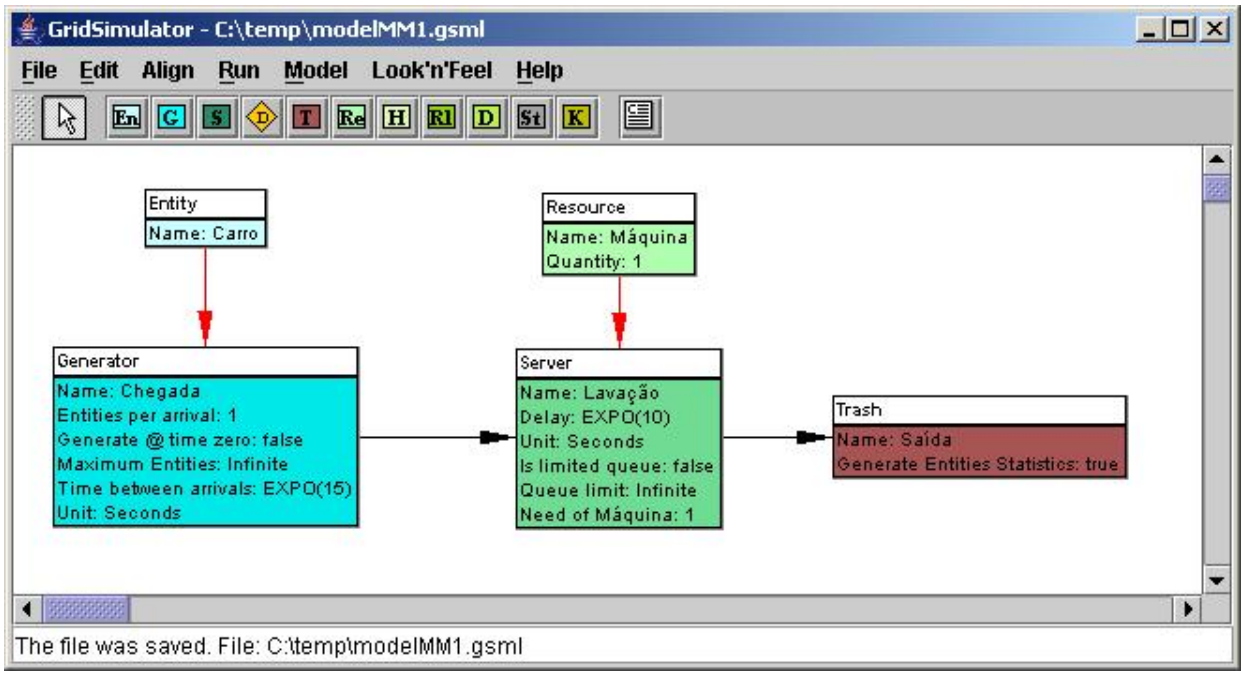

Figura 4.18: Modelo simplificado de um sistema MM1 mostrando todos os componentes

Para construir o modelo, o analista tem algumas funções para auxiliá-lo na forma de apresentação dos componentes, nas alterações das propriedades deles; por exemplo, alterar o tempo entre chegada das entidades no gerador (*generator*). Existe ainda

a função de mostrar ou ocultar os componentes de entidade (*entity*) e recurso (*resource*). A figura 4.18 apresenta um modelo típico de fila MM1 simplificado e todos os componentes usados. O modelo de fila MMn denota que está sendo modelado um sistema em que o tempo entre as chegadas das entidades e o tempo de serviço são exponencialmente distribuídos e existem n servidores. Com o objetivo de apresentar uma idéia mais clara do fluxo lógico do modelo, o analista pode ocultar os componentes Entidade e Recurso (figura 4.19).

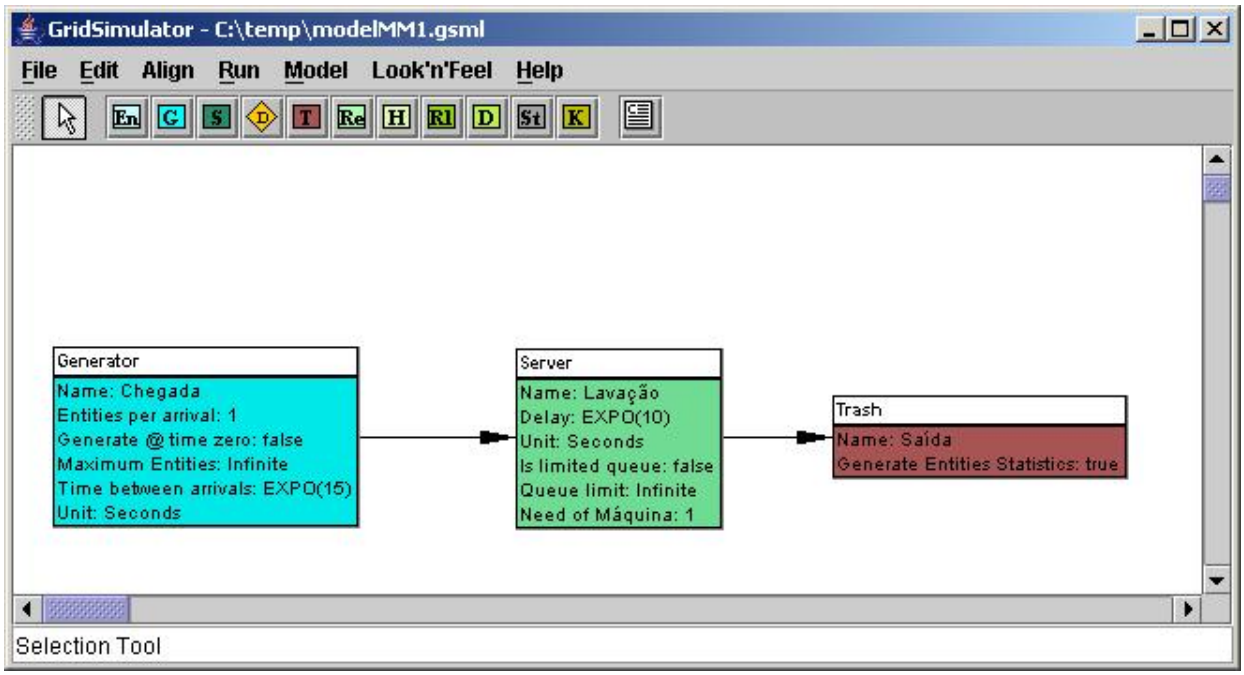

Figura 4.19: Modelo simplificado de um sistema MM1 mostrando apenas os componentes do fluxo lógico

Esta dissertação apresenta uma proposta de visualização dos componentes. Foi tomada como referência a idéia do diagrama de classe da *Unified Modeling Language* (UML). O diagrama de classe da UML, conforme *Object Management Group* (OMG) [Obj 03], é um grafo de elementos classificados e seus vários relacionamentos estáticos. Neste diagrama, os elementos classificados podem representar classes, interfaces, pacotes, relações e até instâncias de objetos. Os elementos são representados pelo

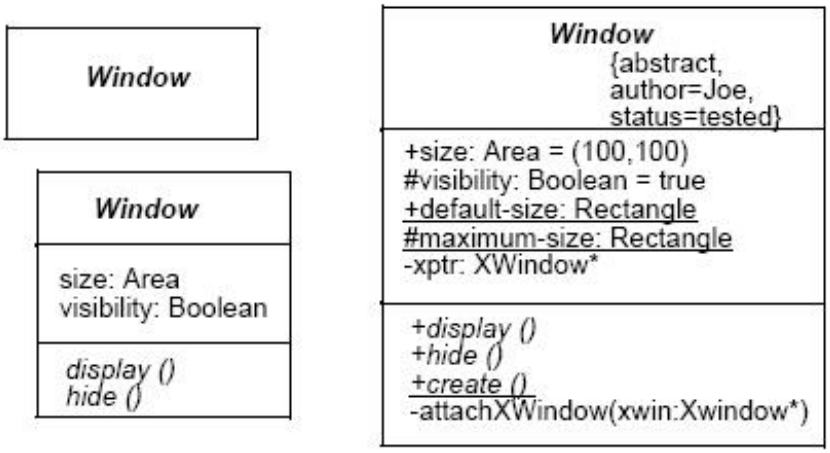

Figura 4.20: Exemplo da representação do elemento classe no diagrama de classe da UML [Obj 03]

desenho de um retângulo dividido em até três regiões (figura 4.20): a região superior (principal) contém o nome do elemento; a região central contém uma lista dos atributos do elemento; e a região inferior contém uma lista dos métodos do elemento.

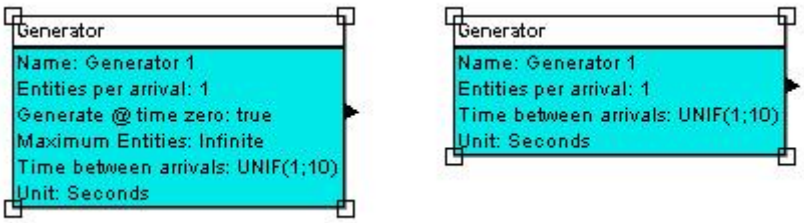

Figura 4.21: Componente *generator* apresentando diferentes propriedades

Tendo como referência o diagrama de classe, a representação dos componentes também utiliza um retângulo, mas com apenas duas regiões: na região superior, de cor branca, está o nome do tipo de componente representado, e na região inferior, da cor que representa o tipo do componente, encontra-se uma lista com as propriedades do respectivo componente, conforme exemplificado na figura 4.21. A utilização desse formato

de representação possibilita ao analista selecionar quais propriedades de um determinado componente devem estar visíveis. Desse modo, o analista pode deixar visíveis apenas as propriedades que ele considere relevante para o modelo a ser construído, conforme mostra a figura 4.21.

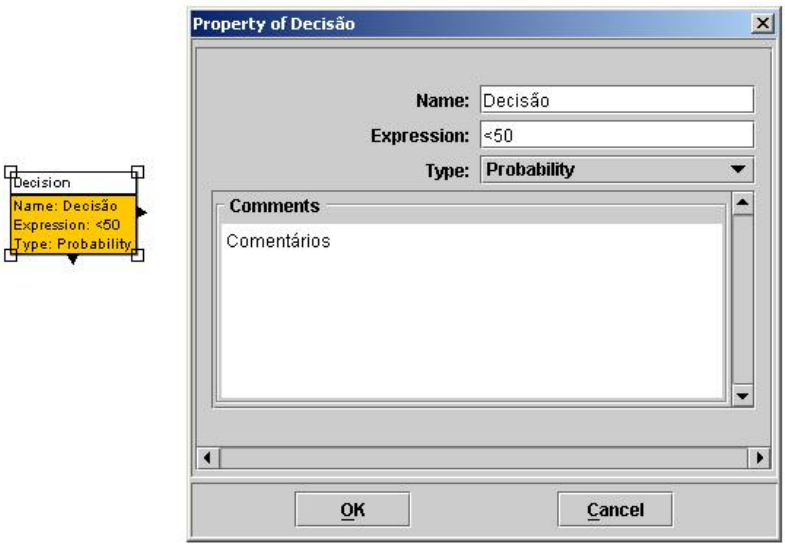

Figura 4.22: Tela de diálogo para alteração das propriedades

Com relação às propriedades de cada componente, elas são alteradas por meio de uma tela de diálogo, apresentada na figura 4.22. Os campos da tela de diálogo são apresentados de acordo com o componente selecionado. A tela carrega as propriedades, seus respectivos editores e, então, as disponibiliza para alteração. Na figura 4.22 há um exemplo da tela de diálogo do componente de decisão (*decision*).

4.2.1 Componentes de modelagem

Componentes são blocos simples que, ao serem agregados entre si, formam um fluxo lógico, modelando um sistema. Esses blocos possuem objetivos distintos. Por exemplo, o componente de decisão realiza um alteração no fluxo dependendo da escolha que ele deve tomar, enviando a entidade para a saída da decisão verdadeira ou para a

saída da decisão falsa. Um componente pode também ser um agregado de outros componentes, como, por exemplo, o componente servidor (*server*). O Editor possui os seguintes componentes para a modelagem:

a) ***Entity***: representa a entidade que percorrerá o modelo seguindo o fluxo lógico;

b) ***Resource***: representa o recurso necessário para realizar alguma ação;

c) ***Generator***: representa o gerador de entidades;

d) ***Hold***: representa a requisição de recursos para liberar a entidade a prosseguir no fluxo;

e) ***Delay***: representa um tempo de espera;

f) ***Release***: representa a liberação dos recursos atrelados às entidades;

g) ***Decision***: representa uma alteração no fluxo de acordo com o resultado da expressão de decisão;

h) ***Set***: representa a alteração de valores das variáveis e/ou dos atributos das entidades;

i) ***Keep***: representa a inclusão de variáveis selecionadas para o relatório final;

j) ***Trash***: representa a saída do fluxo lógico;

k) ***Server***: representa a agregação dos componentes *hold*, *delay* e *release*; e

l) ***Comments***: possibilita colocar comentários no modelo.

Para o desenvolvimento desses componentes, primeiramente foi definida uma classe-base, a qual concentra os métodos comuns de todos os componentes. Como exemplo pode-se citar os métodos para adicionar e retirar atributos/propriedades dos componentes. A figura 4.23 apresenta o diagrama de classe do relacionamento entre a classe-base, denominada *BaseFigure*, e os componentes. Estes apresentam métodos e propriedades em comum. As propriedades são as seguintes: "*id*", nome, comentário. O "*id*" é um valor usado para identificar um determinado componente. Seu valor é único no modelo. O nome é um campo usado pelo analista para identificar um componente

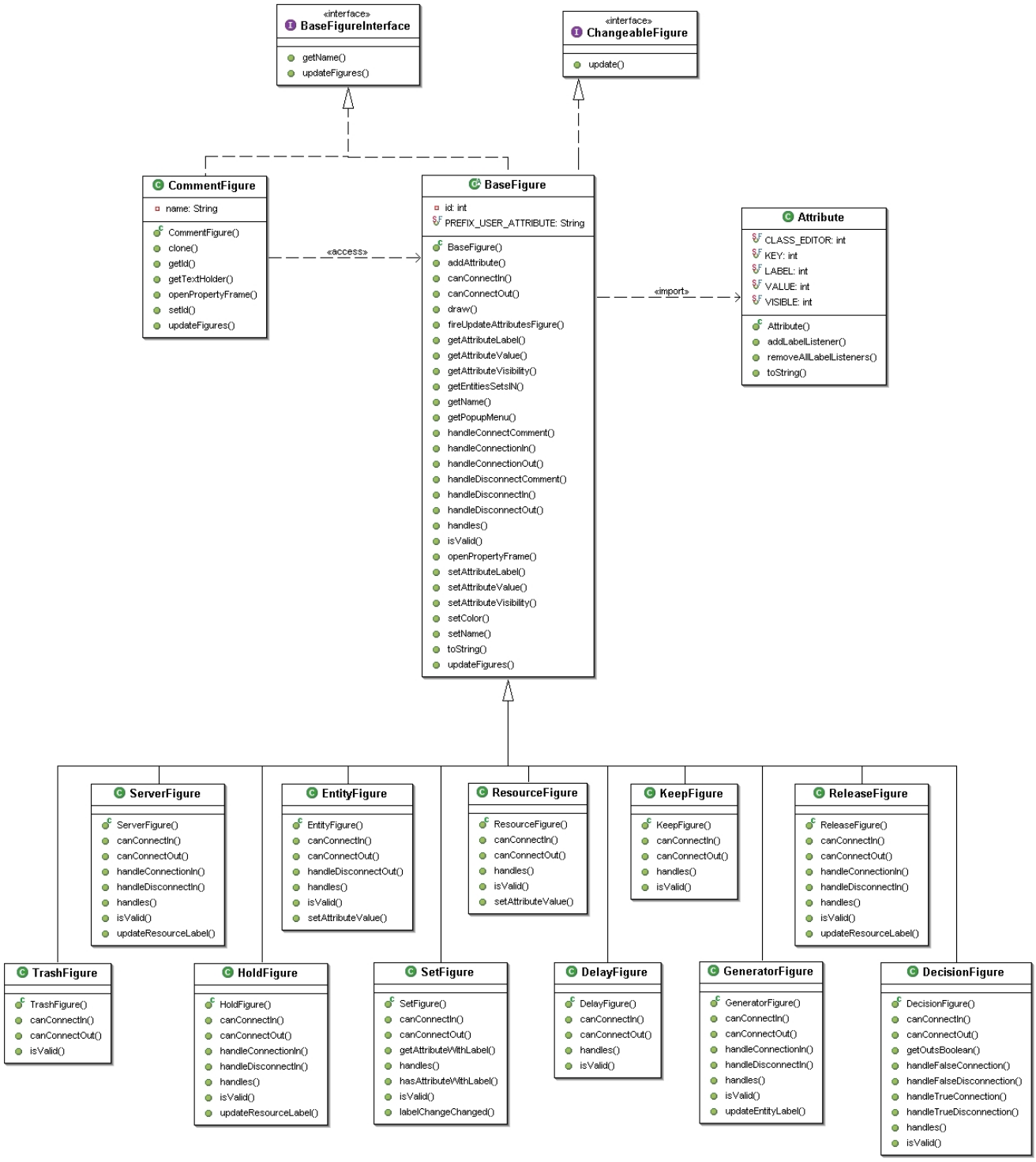

Figura 4.23: Diagrama de classes dos componentes

qualquer. O campo comentário foi disponibilizado para que o analista possa incluir informações que considere necessárias em cada componente do modelo.

Como já foi comentado, o modelo é salvo em arquivo no formato de XML. Usando esse formato pode-se, de maneira organizada, criar a representação dos componentes. O arquivo XML é validado usando as definições do *Document Type Definitions* (DTD) apresentado no apêndice A. Na figura 4.24 está um exemplo de como é armazenado um componente. Nela está exemplificada a representação de um *generator*, o qual tem suas propriedades descritas com seus respectivos valores. Esse *generator* é o componente que está nas figuras 4.18 e 4.19.

```
<component type="package.GeneratorFigure" id="1" xpos="21" ypos="107" visible="true">
  <fixedattribute key="aaname" label="Name" class="package.GSTextField" visible="true" value="Chegada" />
  <fixedattribute key="epa" label="Entities per arrival" class="package.GSTextField" visible="true" value="1" />
  <fixedattribute key="gtz" label="Generate @ time zero" class="package.GSCheckBox" visible="true" value="false" />
  <fixedattribute key="me" label="Maximum Entities" class="package.GSIntegerInfiniteTextField" visible="true" value="Infinite" />
  <fixedattribute key="tba" label="Time between arrivals" class="package.GSTextField" visible="true" value="EXPO(15)" />
  <fixedattribute key="tbaunit" label="Unit" class="package.GSComboTimeUnit" visible="true" value="Seconds" />
  <userattribute key="userattrentityid5" label="Carro's Id" class="package.GSLabel" visible="false" value="5" />
  <fixedattribute key="zzcomment" label="Comments" class="package.GSScrollPaneTextArea" visible="false">
    <value />
   </fixedattribute>
</component>
```

Figura 4.24: Componente *generator* expresso usando XML

Os componentes já foram previamente listados, e foram apresentados seus respectivos objetivos. No entanto, cada um deles difere dos demais nas propriedades, cores e quantidade de relacionamentos. A seguir será apresentado cada um com suas características.

4.2.1.1 *Entity*

Este componente representa a entidade que percorre o fluxo lógico do modelo. Deve sempre estar conectado a pelo menos um *generator* e nenhum outro pode se conectar a ele. Como mostra a figura 4.25, a entidade pode ter de nenhum a vários atributos. A quantidade de atributos é determinada pelo analista. A entidade pode ter mais atributos adicionados a ela no decorrer do modelo. Ela se conecta com o *generator*

por meio de uma seta de cor vermelha. Essa seta tem a cor diferenciada das cores do fluxo (preta e cinza) para não confundir o usuário na leitura do modelo criado.

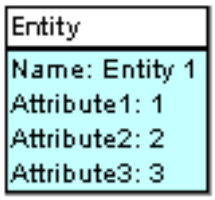

Figura 4.25: Componente *Entity*

4.2.1.2 *Resource*

O *resource* (figura 4.26) representa algum recurso que o analista deseja colocar no modelo. A restrição de conexão deste componente é ele apenas permitir a conexão aos componentes *hold*, *release* e *server*. Nenhum componente pode ser conectado ao recurso. Pode-se representar o compartilhamento de recursos se estes forem conectados a mais de um componente, no caso o *hold* e o *server*.

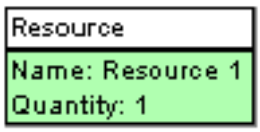

Figura 4.26: Componente *Resource*

A propriedade que deve ser informada para este componente é a quantidade disponível. Essa quantidade é constante, mas o estado desses recursos varia no decorrer do processamento do modelo. O recurso pode estar no estado de ocupado ou ocioso durante a execução do modelo.

4.2.1.3 *Generator*

O *generator* é responsável por gerar as entidades de acordo com seus parâmetros de configuração. Este componente pode se conectar a qualquer outro que aceite a chegada de sua conexão. Ele não pode receber nenhuma conexão além da proveniente do *entity*. Esta conexão que chega não pertence ao fluxo do modelo. As propriedades inerentes apenas ao *generator* estão descritas no quadro 4.1.

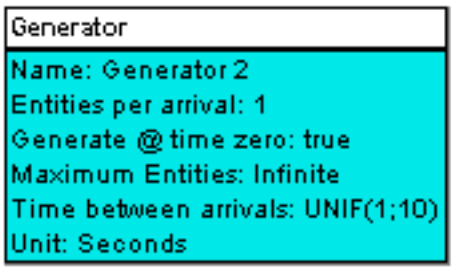

Figura 4.27: Componente *Generator*

Propriedade	**Descrição**	**Valor Padrão**
Entidades por chegada	Número de entidades geradas por chegada	1
Gera no tempo zero	Se gera ou não as entidades no tempo zero	Sim
Número máximo de entidades	Quantidade máxima de entidades que podem ser geradas	Infinito
Tempo entre chegadas	Tempo entre as chegadas das entidades	UNIF(1;10)
Unidade	Unidade do tempo entre chegadas (segundos, minutos, horas e dias)	Segundos

Quadro 4.1: Propriedades do *generator*

4.2.1.4 *Hold*

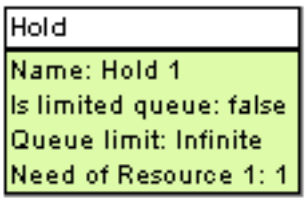

Figura 4.28: Componente *Hold*

O *hold* tem o objetivo de requisitar todos os recursos que nele estão conectados para que possa liberar a entidade para prosseguir no modelo. Como para cada entidade é necessária uma quantidade n de um recurso r, este componente mantém uma fila para as entidades que estão aguardando por sua vez para reter o recurso. Essa fila utiliza a estratégia FIFO (*First In First Out*) para atender as entidades. A fila pode ou não ter um limite máximo. No caso de uma fila com limite, as entidades que chegam e não têm espaço na fila são contadas e descartadas do modelo.

O *hold* tem algumas propriedades que o analista pode modificar: se a fila tem ou não limite; em caso positivo, qual é o limite da fila; e a quantidade de cada recurso conectado a este componente que a entidade irá requisitar para o seu processamento ou serviço, como mostra a figura 4.28.

4.2.1.5 *Delay*

Neste componente (figura 4.29) ocorre uma espera por determinado período, no qual uma entidade deverá aguardar para prosseguir pelo fluxo lógico/físico do modelo. Este se conecta a qualquer componente que aceite chegadas de conexões e recebe conexões de qualquer componente que possa ser conectado a ele. Suas propriedades são: o tempo de espera e a unidade deste tempo. O tempo de espera pode ser uma distribuição estatística combinada ou não com variáveis ou atributos das entidades.

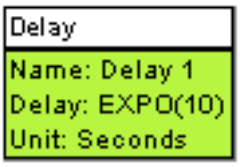

Figura 4.29: Componente *Delay*

4.2.1.6 *Release*

Este componente tem o papel de liberar os recursos requisitados pelo *hold*. Sua estratégia de conexão é igual à do *hold*. Sua única propriedade é a quantidade a ser liberada do recurso a ele conectado, como mostra a figura 4.30.

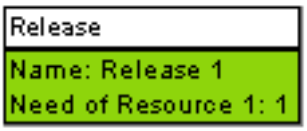

Figura 4.30: Componente *Release*

4.2.1.7 *Decision*

Este é o único componente entre os criados que pode alterar o caminho que a entidade realiza no modelo. Ele pode receber conexões de qualquer componente sem restrição de conexão de saída. Porém, a sua conexão de saída tem duas possibilidades: a saída verdadeira e a saída falsa (seta da cor cinza). Suas propriedades podem ser (figura 4.31): a expressão que avalia qual deve ser a saída da entidade; e o tipo dessa decisão, que pode ser uma decisão probabilística ou condicional. Na decisão probabilística, o analista coloca no campo da expressão algum símbolo de comparação ($==, !=, <, >, <=, >=$) seguido por um valor de 0 a 100 para montar a expressão para a saída verdadeira; por exemplo, "$<= 50$" representa que, se o número sorteado for menor ou igual a 50, a entidade sairá pela saída verdadeira, caso contrário sairá pela falsa. E na decisão condicional o analista informa uma expressão que será avaliada e, de

acordo com seu resultado, enviará a entidade para a saída correspondente. Por exemplo, "$atributo == 1$" representa que, se o atributo da entidade atual for igual a 1, então esta entidade sairá pela saída verdadeira; senão, sairá pela falsa.

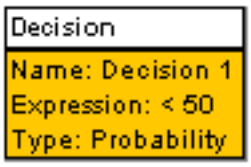

Figura 4.31: Componente *Decision*

4.2.1.8 *Set*

O componente *set* (figura 4.32) é utilizado para alterar os valores das variáveis do modelo e dos atributos das entidades. Sua forma de conexão é semelhante à do *hold*. Ele mantém uma lista de qual atributo/variável deverá ser atualizado. Caso o item que deva ser atualizado não seja uma variável ou um atributo, então este é considerado como um atributo e o cria na entidade com o valor que deveria ser usado na atualização.

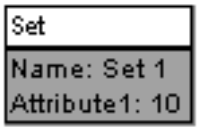

Figura 4.32: Componente *Set*

4.2.1.9 *Keep*

O *keep* (figura 4.33) é utilizado para salvar dados de determinadas variáveis no relatório final. Ele se conecta da mesma forma que o *set*.

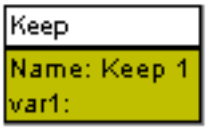

Figura 4.33: Componente *Keep*

4.2.1.10 *Server*

O *server* (figura 4.34) é um agregado de componentes. Ele conjuga os componentes *hold*, *delay* e *release*. Portanto, suas propriedades são as agregações das propriedades desses componentes, e sua forma de conexão se assemelha também com as desses componentes.

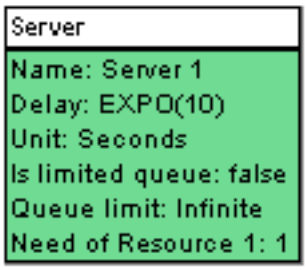

Figura 4.34: Componente *Server*

4.2.1.11 *Trash*

Este componente (figura 4.35) representa o final do caminho que será percorrido pela entidade. Nele são armazenadas as estatísticas das entidades. Ele apenas recebe conexão, mas nenhuma conexão pode partir dele. Sua propriedade é se deve ou não armazenar as estatísticas das entidades.

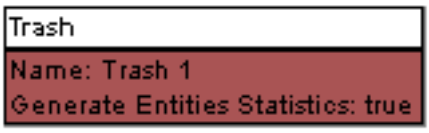

Figura 4.35: Componente *Trash*

4.2.2 Integração com o *grid*

A integração com o *grid* foi realizada utilizando-se o CoG (*Commodity Grid*) para a plataforma Java. O CoG é um *kit*, um conjunto de classes, que provê um acesso facilitado para os recursos do *grid*, permitindo o desenvolvimento mais rápido das aplicações [vL 01]. Um esquema simplificado para submissão do modelo é apresentado na figura 4.37. Os itens de *menu* que realizam a integração estão apresentados na figura 4.36.

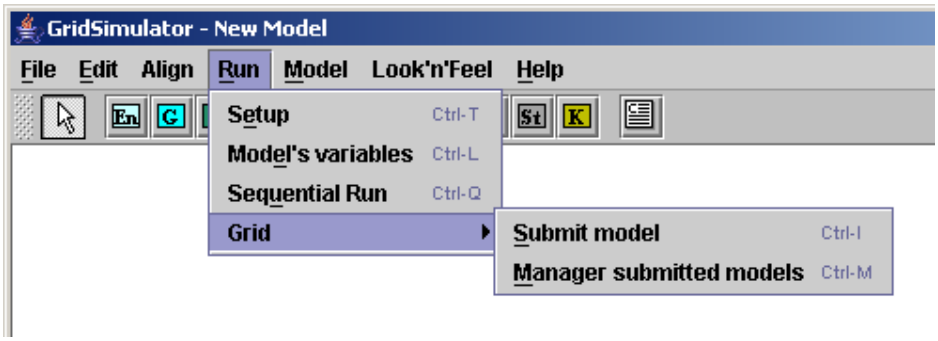

Figura 4.36: Itens de *menu* criados para a integração com o *grid*

No estágio atual do Editor, a consulta para qual nó do *grid* o modelo deve ser enviado será realizada utilizando-se meios externos ao Editor. O meio externo pode ser um possível portal do *grid*, documentação da infra-estrutura ou por meio de contato com os administradores da infra-estrutura. Após conhecer o nó que receberá o modelo para a execução, o analista adquire uma credencial válida para ter acesso aos recursos. Com a credencial obtida, ele realiza o envio do modelo para o nó selecionado. Para exemplificar, foi realizada uma submissão do modelo da figura 4.18, o qual está

configurado de acordo com a figura 4.38.

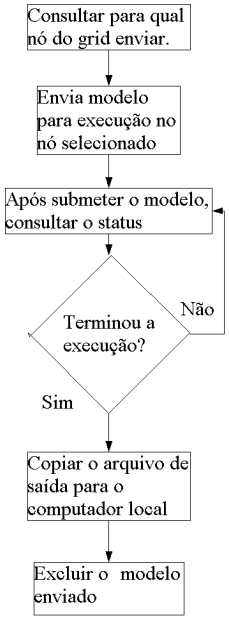

Figura 4.37: Passos simplificados do acompanhamento da submissão do modelo

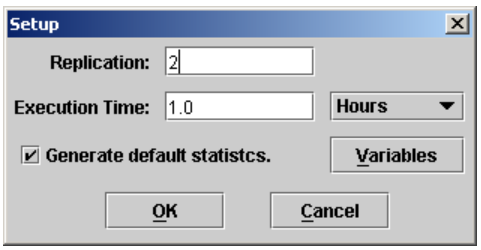

Figura 4.38: Configuração do modelo MM1

A submissão é feita pela tela que está apresentada na figura 4.39. Esta contém os seguintes parâmetros: nome do nó para onde será enviado o modelo (*garopaba.inf.ufsc.br*); e o diretório de destino do modelo (*/home/ufsc*). A submissão do modelo, para ser executado no nó do *grid* especificado, segue a seqüência apresentada pelo diagrama da figura B.1 do apêndice B. Após a submissão, o analista precisa aguardar o fim da simulação do modelo enviado. Este acompanhamento é feito pela tela de gerenciamento de modelos submetidos (figura 4.40).

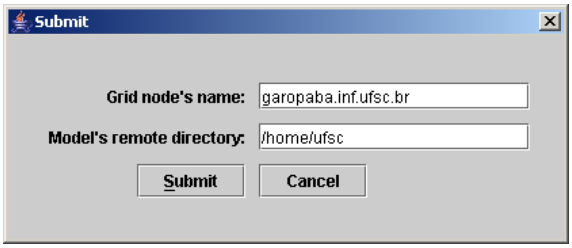

Figura 4.39: Tela para submissão dos modelos

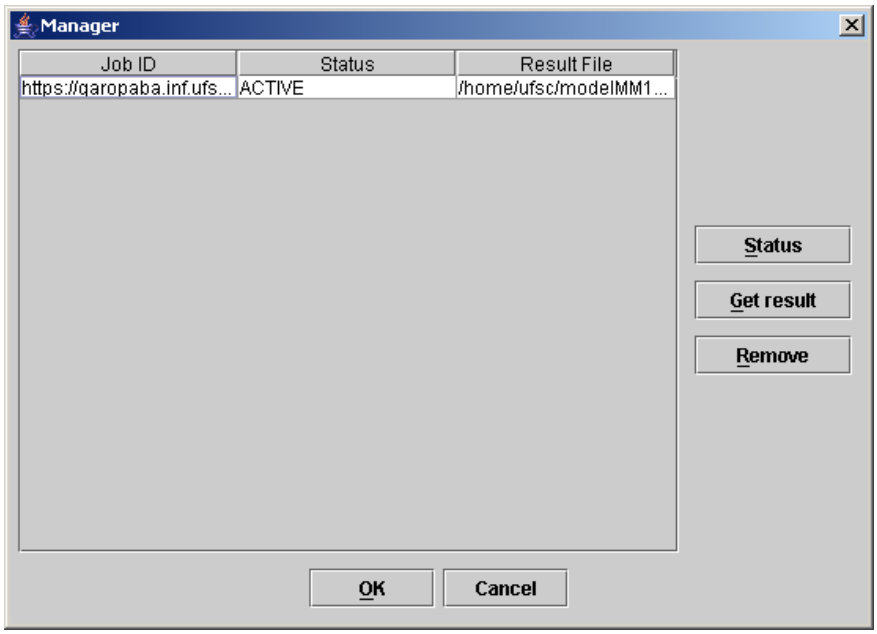

Figura 4.40: Tela de gerenciamento dos modelos submetidos

Para atualizar o *status* da execução basta o analista apertar o botão *Status*, e o Editor se encarrega de realizar a consulta. Por exemplo, a figura 4.41 apresenta o novo *status* do modelo após a atualização, com o *status* alterado para "*DONE*". Esse novo estado de execução do modelo permite ao analista realizar a cópia do arquivo com os resultados usando o botão *Get result*. Para obter sucesso nesta ação o computador do analista deve ser acessível para o nó do *grid*. Finalizando os procedimentos, o analista tem a liberdade de excluir o modelo do nó do *grid* no momento que ele achar necessário, bastando, para isso, selecionar o modelo e apertar o botão *Remove*. O resultado da execução deste exemplo é apresentado na figura 4.42.

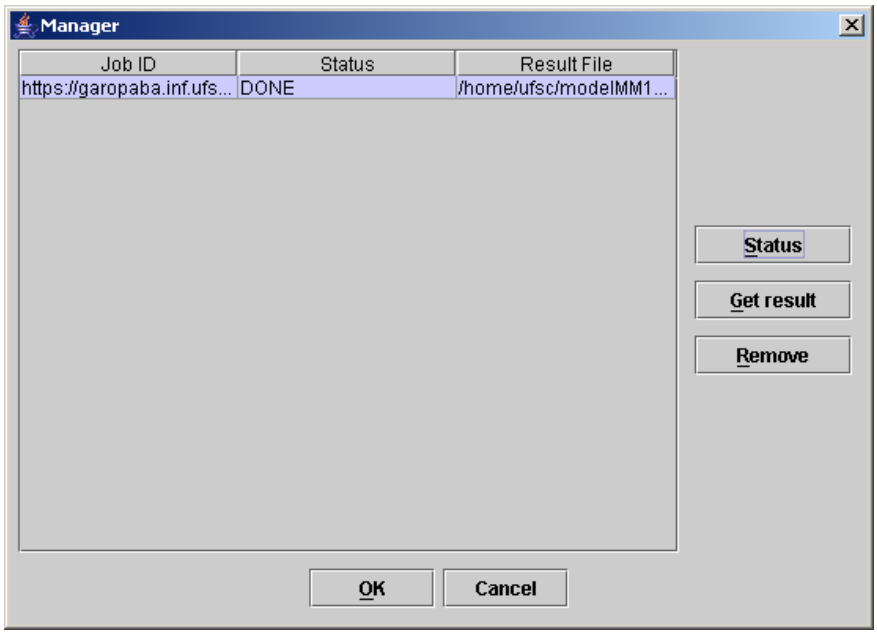

Figura 4.41: Modelo MM1 submetido e concluído

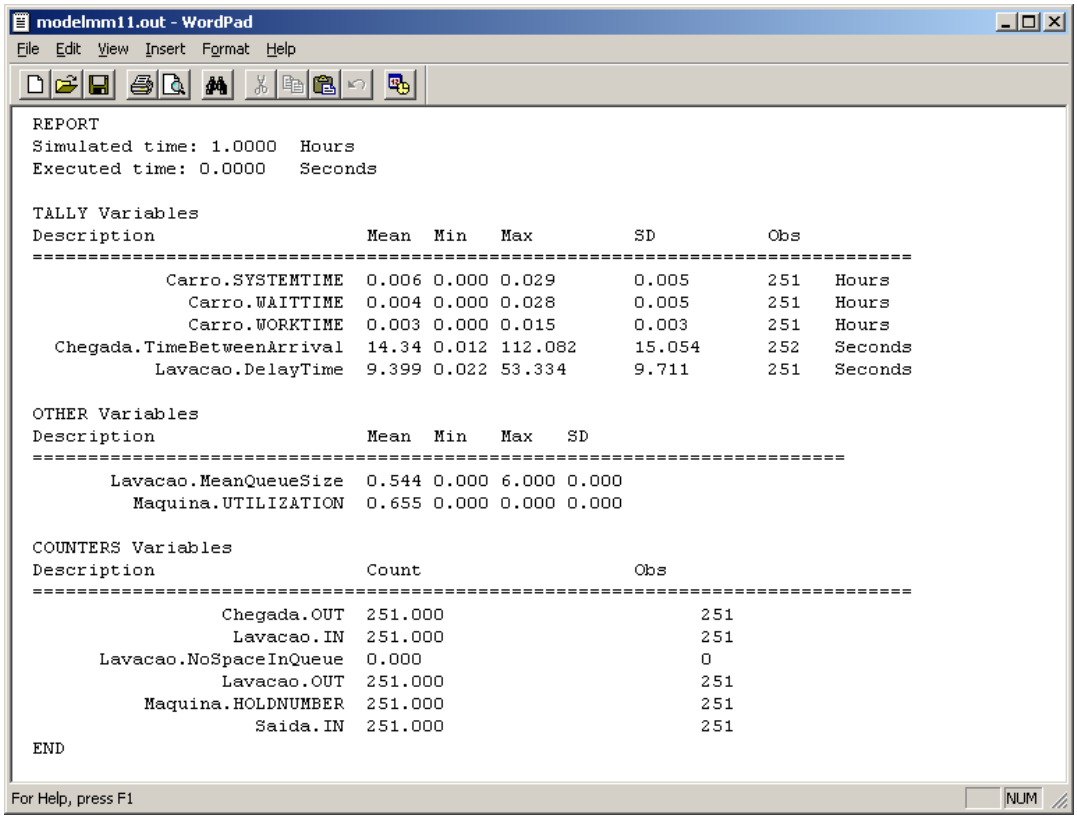

Figura 4.42: Resultado da execução do modelo MM1

4.3 Processador de modelos

O processador de modelos, denominado de Ares, é responsável por interpretar o modelo criado, executá-lo, colher as estatísticas e gerar o relatório final. O modelo é interpretado utilizando-se a biblioteca *Xerces-C* [The 04]. Esta realiza a leitura do arquivo XML gerado pelo Editor e fornece classes para manipular os dados do arquivo. O processador foi desenvolvido utilizando-se a linguagem C++. A escolha foi pela maior velocidade e flexibilidade da linguagem e pelo fato de não necessitar de uma máquina virtual para executar o programa gerado.

O desenvolvimento do Ares foi realizado com base no formalismo DEVS (seção 3.1). Com isso, foram construídos os componentes com suas entradas, saídas e transições de estados. O processamento ocorre conforme mostra o fluxograma na figura 4.43. Primeiramente, são gerados os eventos iniciais de cada componente. Os eventos são mantidos em uma fila ordenada pelo tempo de execução do respectivo evento, sendo o primeiro da fila aquele com o menor tempo. Com os eventos na fila, inicia-se uma iteração para processar os eventos. Retira-se o primeiro evento da fila e realiza-se um teste para verificar se o tempo do evento é maior que o tempo de parada. Em caso positivo, a iteração é encerrada e gera-se o relatório final; em caso negativo, a iteração continua. Realiza-se um novo teste para verificar a consistência do tempo, pois ele não pode ser menor que o tempo atual de execução, pois isso significa que é um tempo no passado e que já deveria ter sido processado. Neste caso também encerra-se a iteração com uma mensagem de erro e gera-se o relatório final até o tempo corrente. Caso os testes sejam malsucedidos, processa-se o evento e depois inicia-se a iteração.

Para a execução dos eventos, os componentes consultam suas propriedades. Estas têm, cada uma, um tipo de dado para armazenar suas informações. Esses tipos podem ser: inteiro; real; booleano; *string*; unidade de tempo; expressão; e expressão booleana. Dos tipos apresentados, apenas os últimos três são tipos de dados diferentes do usual. O tipo unidade de tempo representa se determinada propriedade aceita valores do tipo segundos, minutos, horas ou dias. Já os tipos de dados expressão e expressão booleana serão abordados a seguir.

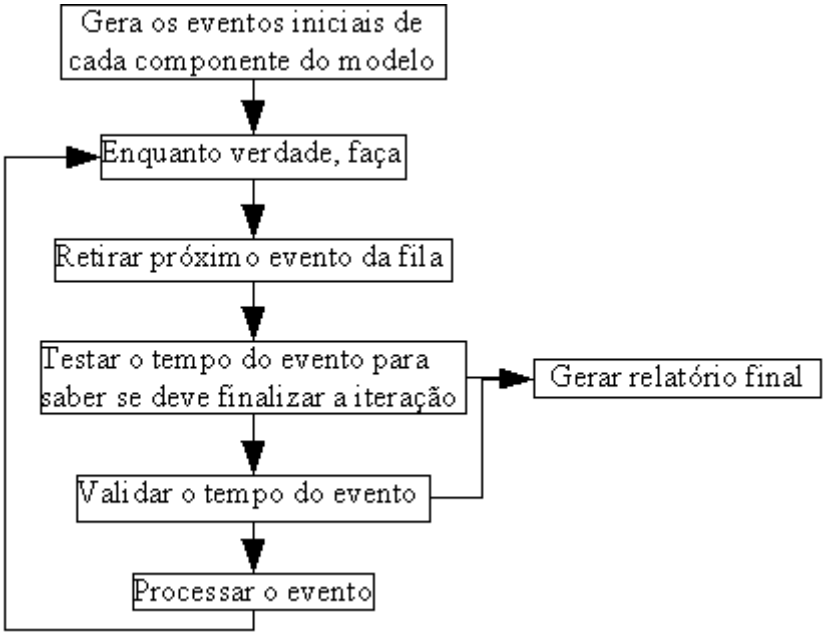

Figura 4.43: Fluxograma da iteração central do Ares

4.3.1 Expressões

Expressões são *strings* que o analista pode construir para que o processador as interprete e gere os resultados apropriados. Nas expressões, o analista pode realizar cálculos usando: as variáveis do modelo; os atributos das entidades; símbolos suportados; valores constantes; distribuições estatísticas; e as quatro operações básicas $(+, -, /, *)$. A ordem de avaliação das operações é multiplicação, divisão, soma e subtração. O resultado da avaliação gera um valor real, mas se na expressão estiver presente o prefixo "*(int)*" gera um valor inteiro.

Para avaliar melhor as partes de uma expressão, segue uma lista com explicações de cada uma.

- **Símbolos suportados** - na versão atual do processador, existe apenas o símbolo *CTime* (*Current Time*). Usando este símbolo o analista tem acesso ao valor corrente do tempo. A unidade deste é a mesma que a unidade do modelo.

- **Variáveis** - são também expressões. O analista pode utilizar as variáveis para me-

lhor parametrizar o modelo em estudo. Cabe destacar que, com este tipo de utilização, elas podem conter relações entre outras variáveis no modelo. Um risco de se usarem variáveis em expressões de outras variáveis é que pode ocorrer recursão infinita e bloquear-se a execução do modelo. Esta recursão pode ocorrer se pelo menos duas variáveis forem dependentes uma da outra. No entanto, o uso de variáveis permite uma grande flexibilização ao analista na criação de modelos diversos.

- **Atributos** - são valores reais. Eles são dados pertinentes às entidades. Todas as entidades podem ter os mesmos atributos, mas os mesmos atributos podem conter diferentes valores.

- **Distribuições estatísticas** - são funções geradoras de números aleatórios de acordo com determinada distribuição probabilística. Valores constantes como 5 ou $17,34$ são considerados distribuições constantes, pois sempre geram o mesmo número. As distribuições suportadas são apresentadas no quadro 4.2. Neste estão descritos os nomes, a abreviatura, a quantidade de parâmetros da função e um exemplo. Para criar uma função de distribuição usam-se as quatro primeiras letras do nome da distribuição e colocam-se os parâmetros entre parênteses separando-os com o caracter ";". Mais informações sobre as distribuições probabilísticas podem ser obtidas em [LAW 92].

Para realizar a avaliação, é construída recursivamente uma estrutura de árvore que representa a expressão informada. Essa estrutura tem em seus nós-folhas os operandos, e nos nós internos, os operadores. Para exemplificar a avaliação, as seguintes expressões foram criadas: "$NORM(15;2) * 100/UNIF(10;15) - 5 + POIS(15)$" e "atributo+variável$/100$". Essas expressões, quando são criadas no processador, geram as árvores das figuras 4.44 e 4.45, respectivamente.

4.3.1.1 Expressões booleanas

As expressões booleanas são comparações entre as expressões apresentadas na seção 4.3.1. As comparações podem ser: $==$ (igualdade), $!=$ (diferença),

Nome	Abreviatura	Número de parâmetros	Exemplo
Uniforme	UNIF	2	UNIF(12;34)
Normal	NORM	2	NORM(10;2)
Erlang	ERLA	2	ERLA(4;15.0)
Beta	BETA	2	BETA(3;40)
Gamma	GAMM	2	GAMM(15;50)
LogNormal	LOGN	2	LOGN(15;3)
Weibull	WEIB	2	WEIB(15;3)
Exponencial	EXPO	1	EXPO(12)
Poisson	POIS	1	POIS(15)
Triangular	TRIA	3	TRIA(50;110;200)
Constante	-	1	145

Quadro 4.2: Distribuições estatísticas

$>$ (maior que), $<$ (menor que), $>=$ (maior ou igual que) e $<=$ (menor ou igual que). Por exemplo, pode-se ter a seguinte expressão booleana: atributo$<=$variável$+10$. Nesse exemplo, será avaliada a primeira expressão, depois a segunda e, após, com seus respectivos resultados, será feita a comparação desejada ($<=$). Este tipo de expressão é normalmente usado pelo componente de decisão (seção 4.2.1.7).

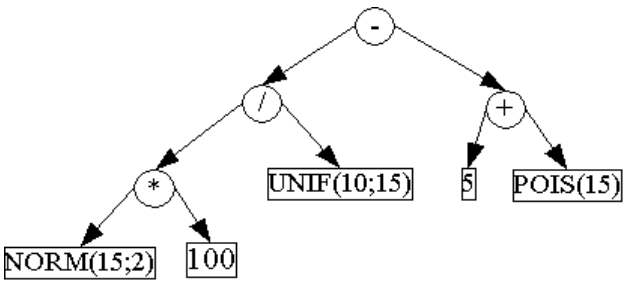

Figura 4.44: Árvore de avaliação do exemplo 1

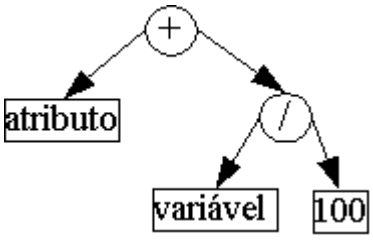

Figura 4.45: Árvore de avaliação do exemplo 2

4.3.2 Componentes de execução

Além das especificações dos componentes fornecidas na seção 4.2.1, há também as estatísticas que cada componente pode gerar, sendo elas coletadas através de componentes específicos para esse fim ou geradas automaticamente (estatísticas padrão). Elas estão listadas por componentes no quadro 4.3, onde se têm os nomes dos componentes, as estatísticas geradas por eles, se elas são geradas por padrão e se elas são condicionais, pois existe uma opção para ativá-las ou desativá-las do relatório final.

Componente	**Estatísticas**	**Padrão**	**Condicional**
Entity	-	-	-
Resource	Utilização	Sim	Sim
	Quantidade de solicitações (hold)	Sim	Sim
Generator	Contador de saídas	Sim	Sim
	Estatística do tempo entre chegadas	Sim	Sim
Hold	Contador de entradas e saídas	Sim	Sim
	Contador de entidades descartadas por falta de espaço na fila	Sim	Sim
	Tamanho médio da fila	Sim	Sim
	Tempo médio de espera na fila	Sim	Sim
Delay	Contador de entradas e saídas	Sim	Sim
	Estatística do tempo de espera	Sim	Sim
Release	Contador de entradas e saídas	Sim	Sim
Decision	Contador de entradas	Sim	Sim
	Contador de saídas verdadeiras	Sim	Sim
	Contador de saídas falsas	Sim	Sim
Set	-	-	-
Keep	Estatísticas das variáveis escolhidas	Não	Não
Trash	Contador de entradas	Sim	Sim

Quadro 4.3: Estatísticas coletadas pelos componentes

Componente	Estatísticas	Padrão	Condicional
	Tempo da entidade no sistema	Não	Sim
	Tempo da entidade em espera	Não	Sim
	Tempo da entidade em trabalho com recurso	Não	Sim
Server	Contador de entradas e saídas	Sim	Sim
	Contador de entidades descartadas por falta de espaço na fila	Sim	Sim
	Tamanho médio da fila	Sim	Sim
	Tempo médio de espera na fila	Sim	Sim
	Estatística do tempo de espera	Sim	Sim

Quadro 4.3: Estatísticas coletadas pelos componentes

4.4 Resumo

Este capítulo apresentou a especificação e, com base nela, o programa desenvolvido nesta dissertação. Informações de como usar o GridSimulator Editor para criar modelos foram apresentadas. Foi apresentado, também, o modo como o Editor pode se conectar ao ambiente *grid*. Para a criação dos modelos pelo programa foram desenvolvidos vários componentes. Interligando-os e alterando suas propriedades, pode-se montar um modelo de um sistema. Para validar esse programa, foi realizado um experimento que está descrito no próximo capítulo.

Capítulo 5

Análise dos resultados

Este capítulo apresenta os resultados de testes realizados com o GridSimulator. Tais testes visam a validar os resultados obtidos pelo simulador, bem como verificar seu desempenho quando executado em ambiente *grid*. O primeiro experimento contrapõe os resultados obtidos pelo GridSimulator com aqueles providos por outro programa de simulação executando um modelo equivalente. Para avaliar o desempenho obtido pelo simulador, medido pelo tempo total de simulação, um segundo experimento foi realizado. Neste, comparam-se os tempos totais de simulação do modelo quando executado em um ou dois nós do *grid*.

5.1 Modelo

O modelo é de um sistema simplificado Cliente/Servidor. Este sistema é descrito em Freitas [dFF 01]:

> Em uma rede local estão conectados computadores de operadores de um serviço de atendimento a clientes e um computador (servidor), no qual são processadas operações sobre um banco de dados, o qual contém uma base de dados armazenada em um único disco.

> Neste sistema, as estações de trabalho são operadas por funcionários de uma empresa que presta informações por telefone. Como esta empresa é muito solicitada, considera-se que os operadores estão sempre ocupados durante o tempo de simulação. Ao receber uma pergunta via telefone, o operador submete uma requisição ao servidor de base de dados via rede local. Esta requisição tem sua sintaxe e semântica verificada no processador local antes de ser submetido à rede de transmissão. Se OK, a requisição é então enviada ao servidor.
> O servidor recebe requisições provenientes de todas as estações e as submete ao sistema gerenciador de banco de dados (SGBD). O SGBD é um conjunto de programas executado no servidor. Após a consulta, o servidor envia a resposta para a estação que originou o comando. A resposta trafega pela rede e chega ao operador. O operador examina a resposta obtida e responde a pergunta do cliente da empresa via telefone. Após um tempo [...] o operador (devido a um novo questionamento do cliente na linha ou devido a sua própria necessidade de informações) envia um novo comando ao servidor. Este processo se repete em todas as estações.

Para a realização dos testes, esse sistema foi modelado no Arena® e no Editor, como mostram as figuras 5.1 e 5.2, respectivamente.

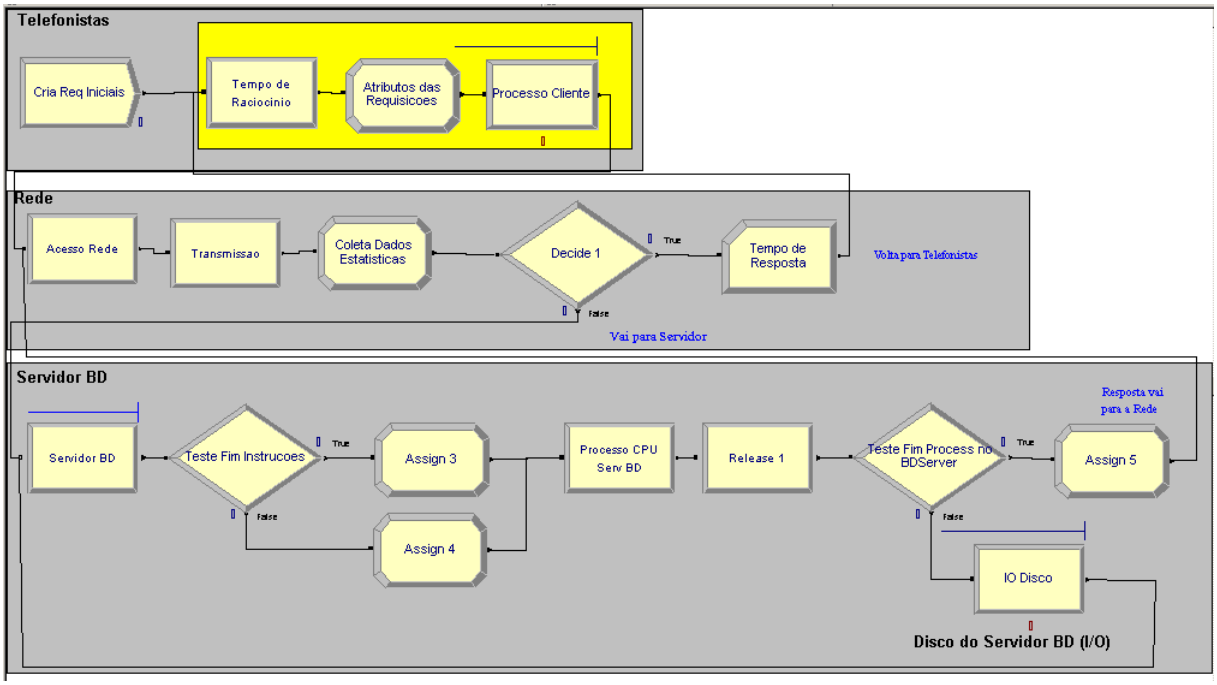

Figura 5.1: Modelo no Arena®

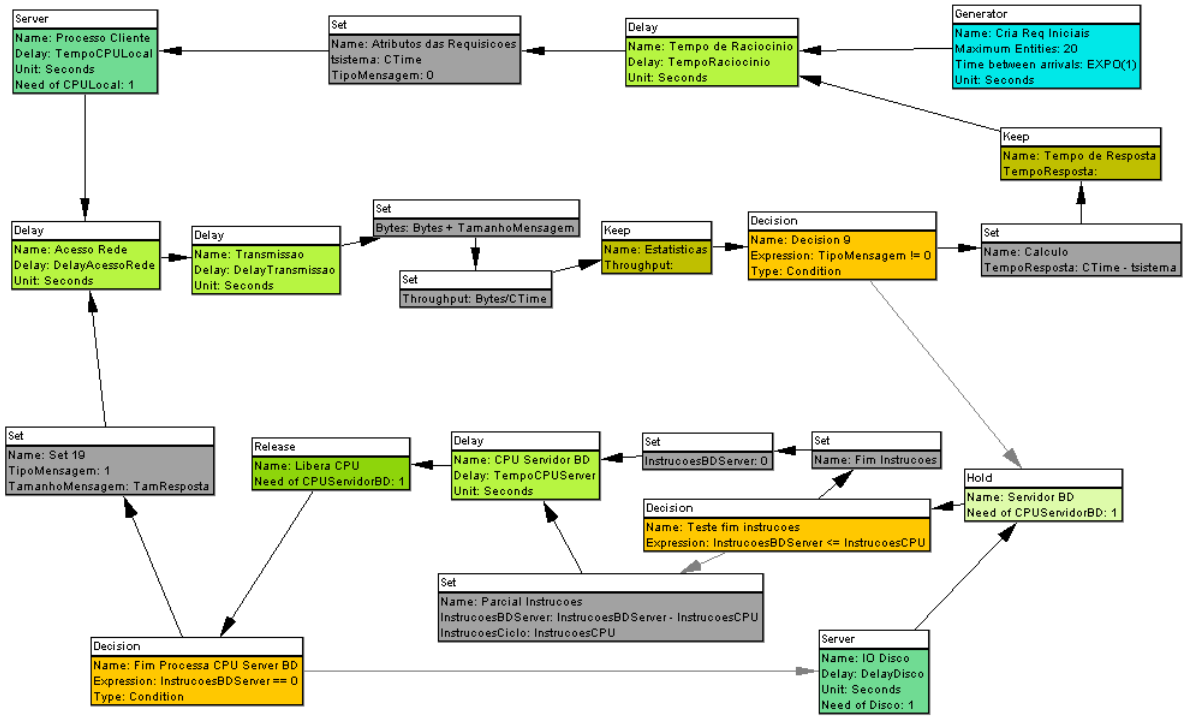

Figura 5.2: Modelo no GridSimulator Editor

5.2 Ambiente de teste

Para executar os modelos criados nos programas foram utilizados um ambiente *grid* e um computador portátil (cliente). Para criar o ambiente *grid* foram utilizados o *middleware globus toolkit* e três computadores. As configurações dos computadores do ambiente *grid* (figura 5.3) estão apresentadas no quadro 5.1. Nele está descrita a configuração de cada computador envolvido nos testes.

Nome	**Memória (Mb)**	**Processador (GHz)**	**Sistema Operacional**
Cliente	512	1,1	WindowsXP
Garopaba	512	1,6	Linux (Red Hat9)
Garopaba2	256	1,6	Linux (Red Hat9)
nodo1	256	1,6	Linux (Red Hat9)

Quadro 5.1: Configurações dos computadores

O modelo criado no Arena® foi executado no cliente, e o modelo criado no GridSimulator Editor foi executado no ambiente *grid* através da integração nele existente (seção 4.2.2). Foram utilizados os dois nós do *grid*, cada um responsável por simular a metade de cada simulação (50 replicações).

5.3 Experimentos

Visando a testar os programas desenvolvidos nesta dissertação, dois experimentos foram realizados. Os experimentos executaram ao todo 100 replicações do modelo. Cada replicação processa o modelo por um determinado período. Esses experimentos serão vistos em seguida.

5.3.1 Experimento para validação dos resultados

O objetivo deste experimento é o de validar os valores obtidos das variáveis de resposta que o GridSimulator apresenta ao final da simulação. O modelo foi

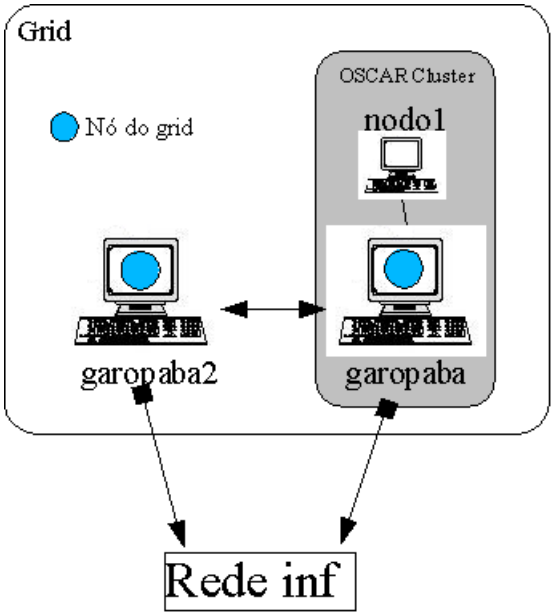

Figura 5.3: Ambiente *grid* do PerformanceLab

simulado com as configurações apresentadas no quadro 5.2. Nesta apresentam-se os parâmetros alterados para cada execução. As variáveis de resposta (VsR) analisadas estão no quadro 5.3. Nesta apresentam-se as descrições e as unidades de cada uma. Foi escolhido o tempo de resposta para avaliar o desempenho do sistema para atender a uma consulta do operador. As outras variáveis são referentes ao desempenho do disco no servidor, pois, conforme Freitas [dFF 01], ali se encontra um gargalo do sistema. Para identificar as combinações de programa, VsR e configuração, utilizou-se a seguinte notação: LLVnCn, onde LL identifica a origem dos dados e pode ser AR, se originados do Arena, ou GS, se originados do GridSimulator. O n identifica a variável e a configuração, variando de 1 a 4.

Uma vez executadas as simulações, foram colhidos os valores para cada uma das quatro VsR de cada uma das configurações. Foi gerado um sumário dos valores com os seguintes campos: N (tamanho da amostra), Média, Intervalo de confiança de 95%, Mínimo, Máximo e o Desvio padrão. O sumário foi construído com a média dos valores obtidos em cada variável de cada replicação para cada configuração. Na figura 5.4 está apresentado o sumário com os resultados dos cálculos realizados com os valores

Configuração	Entidades geradas	Tempo simulado (seg)
C1	20	900
C2	20	9.000
C3	50	900
C4	50	9.000

Quadro 5.2: Configurações utilizadas nos testes

Sigla	Variável de resposta	Unidade
V1	Tempo de espera na fila do disco	Segundos
V2	Tempo de resposta	Segundos
V3	Utilização do disco	Porcentagem ($*100$)
V4	Tamanho médio da fila do disco	Entidades

Quadro 5.3: Variáveis de resposta analisadas

gerados pelo GridSimulator. Já a figura 5.5 apresenta o sumário com os resultados dos cálculos realizados com os valores gerados pelo Arena®.

Variable	N	Média	Confiança -95,000%	Confiança +95,000%	Mínimo	Máximo	Desv Padrão
GSV1C1	100	0,04646	0,04577	0,04715	0,03900	0,05400	0,003457
GSV2C1	100	0,91090	0,90403	0,91777	0,82900	0,98400	0,034631
GSV3C1	100	0,63362	0,63089	0,63635	0,59800	0,66100	0,013770
GSV4C1	100	0,80046	0,78640	0,81452	0,62900	0,96200	0,070876
GSV1C3	100	0,71898	0,71495	0,72301	0,66900	0,76600	0,020331
GSV2C3	100	7,29236	7,24994	7,33478	6,76100	7,82700	0,213776
GSV3C3	100	0,98176	0,98088	0,98264	0,96600	0,99100	0,004420
GSV4C3	100	19,18969	19,08070	19,29868	17,78100	20,49500	0,549288
GSV1C2	100	0,04661	0,04640	0,04682	0,04400	0,05000	0,001043
GSV2C2	100	0,91235	0,91036	0,91434	0,88500	0,94000	0,010031
GSV3C2	100	0,64007	0,63912	0,64102	0,62800	0,65000	0,004791
GSV4C2	100	0,81042	0,80605	0,81479	0,75200	0,87000	0,022002
GSV1C4	100	0,73583	0,73443	0,73723	0,71800	0,75000	0,007051
GSV2C4	100	7,46147	7,44728	7,47566	7,28300	7,62000	0,071513
GSV3C4	100	0,99806	0,99794	0,99818	0,99700	0,99900	0,000583
GSV4C4	100	19,95760	19,91952	19,99568	19,47000	20,35000	0,191918

Figura 5.4: Sumário dos valores gerados pelo GridSimulator

As médias das médias das variáveis de resposta obtidas pelos dois programas apresentam uma diferença. Além disso, os perfis delas (valores das médias entre

Variable	N	Média	Confiança -95,000%	Confiança +95,000%	Mínimo	Máximo	Desv Padrão
ARV1C1	100	0,04622	0,04555	0,04690	0,03855	0,05542	0,003397
ARV2C1	100	0,90897	0,90228	0,91566	0,84023	0,99639	0,033726
ARV3C1	100	0,63370	0,63114	0,63626	0,59949	0,66289	0,012901
ARV4C1	100	0,79675	0,78295	0,81055	0,64825	0,99560	0,069552
ARV1C3	100	0,71684	0,71278	0,72090	0,67389	0,77611	0,020465
ARV2C3	100	7,27450	7,23239	7,31662	6,79747	7,86091	0,212246
ARV3C3	100	0,98161	0,98073	0,98248	0,97110	0,99384	0,004402
ARV4C3	100	19,12959	19,01879	19,24040	18,00474	20,61867	0,558420
ARV1C2	100	0,04665	0,04644	0,04685	0,04379	0,04988	0,001043
ARV2C2	100	0,91325	0,91116	0,91535	0,88556	0,94864	0,010562
ARV3C2	100	0,63990	0,63902	0,64077	0,62679	0,65000	0,004402
ARV4C2	100	0,81116	0,80686	0,81546	0,75287	0,87370	0,021682
ARV1C4	100	0,73541	0,73412	0,73670	0,71907	0,75059	0,006501
ARV2C4	100	7,45488	7,44131	7,46844	7,29811	7,61785	0,068346
ARV3C4	100	0,99811	0,99803	0,99820	0,99711	0,99937	0,000439
ARV4C4	100	19,94727	19,91196	19,98258	19,49620	20,36074	0,177948

Figura 5.5: Sumário dos valores gerados pelo Arena®

o valor mínimo e o máximo resultante) estão em um intervalo parecido. Essa comparação está apresentada no apêndice C, com gráficos com os valores obtidos de cada variável pelos dois programas. Em adição, tem-se calculado o intervalo de confiança de 95% da média de cada variável, com o qual se pode traçar gráficos para comparação. Para exemplificar essa comparação, foram gerados os gráficos (figuras 5.6 e 5.7) da variável de resposta 2 (V2) nas configurações 1 e 2. Com essa comparação, mostra-se que os intervalos de confiança para a média se sobrepõem, tendo-se, então, a semelhança estatística dos valores obtidos pelo GridSimulator com os providos pelo Arena®. Esse comportamento se repete na comparação de todas as VsR analisadas.

Entretanto, essa validação pode ser apenas confirmada para o modelo deste experimento. Para obter uma validação mais completa e genérica dos resultados, mais experimentos com diferentes modelos devem ser realizados.

5.3.2 Experimento para avaliação de desempenho

Este experimento foi realizado para comparar os tempos de simulação obtidos quando se simula usando um ou dois nós do ambiente *grid*. Foi utilizado o mesmo modelo do experimento anterior com a configuração C4 do quadro 5.2. O experimento

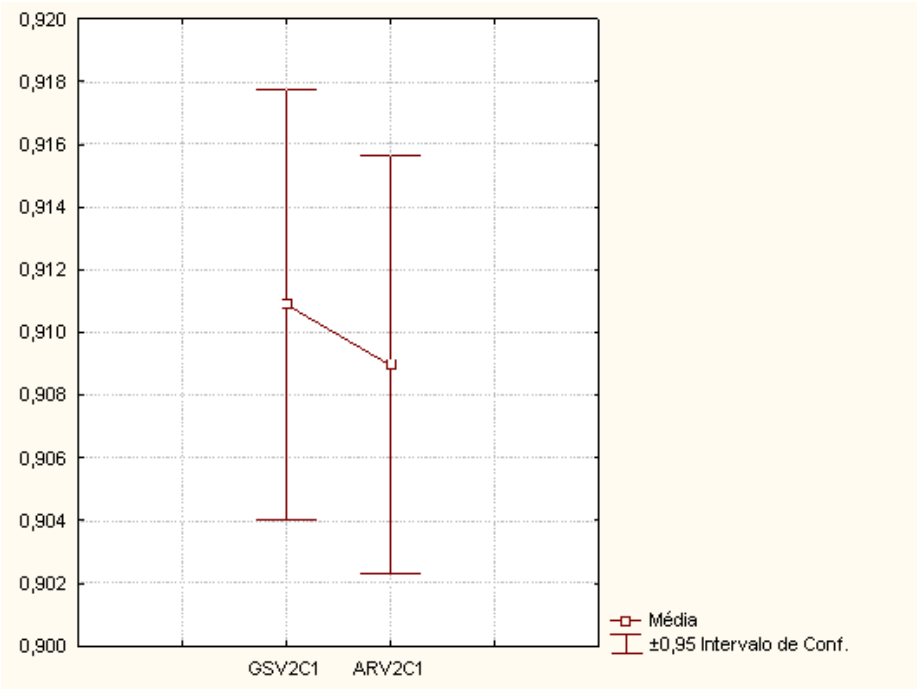

Figura 5.6: Gráfico de comparação da variável de resposta 2 na configuração 1

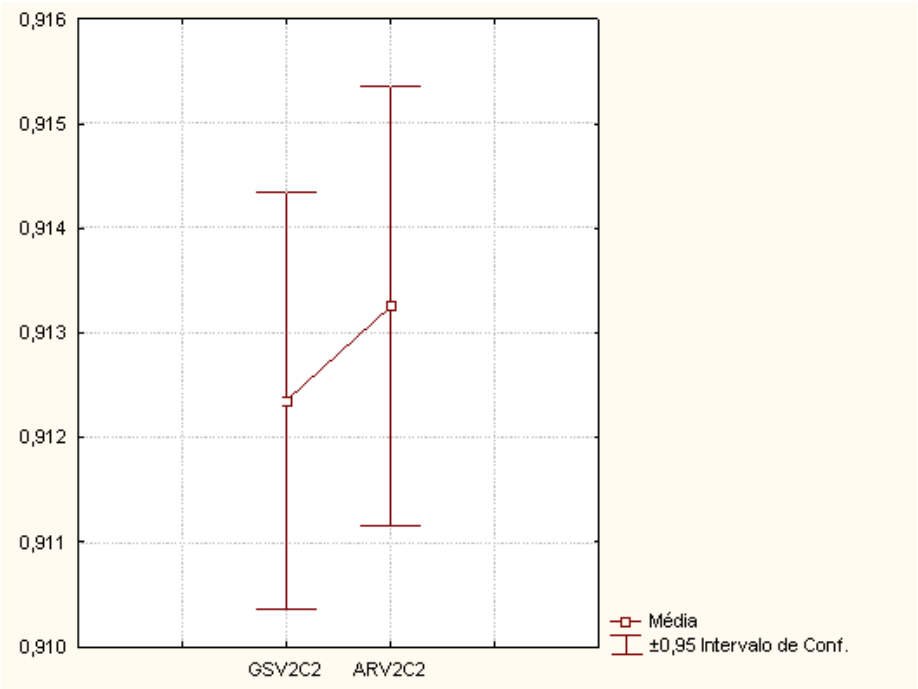

Figura 5.7: Gráfico de comparação da variável de resposta 2 na configuração 2

foi realizado e o seu resultado está descrito no quadro 5.4.

Simulação	**Número de nós**	**Tempo (min)**
A	1	137,63
B	2	68,37

Quadro 5.4: Comparação de tempo de simulação

Na simulação A, realizada com um nó, o tempo de simulação foi de 137,63 minutos. Na simulação B, onde foram usados dois nós, o tempo foi menor que a metade do tempo da simulação A. Essa comparação apresenta a vantagem na simulação quando utilizado mais de um nó do *grid*. Entretanto, ele não afirma que o tempo diminua de modo linear à medida que mais nós forem adicionados à simulação, pois o menor tempo possível de simulação deste experimento é, no mínimo, o tempo de uma única replicação.

5.4 Resumo

Neste capítulo foram apresentados alguns resultados de um experimento desenvolvido com o intuito de validar os valores gerados pelo GridSimulator, comparando-os com os valores gerados pelo programa Arena®. Foi também descrito o modelo utilizado para a comparação e apresentados os resultados colhidos com alguns comentários sobre eles. No entanto, conclusões desta dissertação serão delineadas no próximo capítulo.

Capítulo 6

Conclusão

Nesta dissertação abordaram-se fundamentos de computação de alto desempenho e de simulação. Foram vistos dois ambientes que provêem alto desempenho: *cluster* e *grid*. A popularidade destes ambientes se deve, principalmente, ao declínio do custo de aquisição de computadores usados em sua montagem. Além disso, esses computadores estão ficando cada vez mais potentes.

Na abordagem da simulação, foram apresentados temas fundamentais para sua compreensão. Entre esses temas destacam-se: os mecanismos de avanço no tempo; os modelos; e a simulação discreta. Estes delinearam o desenvolvimento do processador de modelos criado nesta dissertação.

Para atingir os objetivos propostos, foi desenvolvida uma ferramenta para a edição de modelos de simulação discreta e outra complementar a essa, que executa os modelos. A comunicação entre essas ferramentas ocorre com o uso do XML. Ele foi utilizado por se tratar de uma linguagem de marcação que pode ser lida em qualquer linguagem que tiver um "*parser*" disponível para realizar a leitura. Essa separação possibilita que outros programas possam ser feitos para complementar a atual ferramenta; por exemplo, pode-se criar um otimizador dos modelos criado pelo Editor ou criar um outro editor de modelos para apenas utilizar o processador desta dissertação. O Editor, do GridSimulator, provê ao analista uma quantidade de componentes para possibilitar a criação dos modelos. A construção acontece com a interligação desses componentes e

com a alteração das propriedades de cada componente do modelo. Com a flexibilidade dos componentes essa ferramenta tem a possibilidade de gerar modelos de diversos sistemas. Para mostrar a maleabilidade dela, nesta dissertação foi descrito um modelo de fila simples (MM1) e um modelo de um sistema Cliente/Servidor. O Editor foi desenvolvido com o paradigma de orientação a objeto. Este supriu todas as necessidades de desenvolvimento do Editor, mas esse tipo de aplicação pode utilizar outros paradigmas. Um deles é a programação orientada a aspectos, pois existem algumas características que "atravessam" determinadas classes, principalmente com relação à coleta de estatísticas.

Para melhor apresentar os componentes, a representação deles foi inspirada no diagrama de classes da UML. Com esse tipo de representação o analista pode consultar os valores das propriedades sem a necessidade de outra ação do programa, como apresentá-las em uma tela de diálogo. Essa representação permitiu que fossem desenvolvidos métodos para ocultar e mostrar as propriedades, permitindo ao analista escolher qual deve estar visível no componente do modelo.

Os modelos podem ser executados de modo seqüencial ou em um ambiente de *grid*, o qual é o foco principal desta dissertação. Para facilitar o envio do modelo para execução no *grid*, foram criadas algumas funcionalidades no Editor que possibilitaram a integração com o *grid*. Nessa integração foram encontradas algumas dificuldades no decorrer de seu desenvolvimento. A principal delas foi a geração da credencial, pois se ela fosse gerada pelo Editor evitaria a necessidade do uso de outro programa. O problema foi que a biblioteca utilizada para a geração da credencial apresenta alguns problemas quando usada no sistema operacional Windows XP®. Alguns contatos foram feitos com a equipe responsável pelo desenvolvimento da biblioteca, mas o problema persiste.

Para processar os modelos desenvolvidos no Editor, foi criado um programa para interpretá-lo e executá-lo. Este programa, ao finalizar a execução, gera um relatório com as estatísticas coletadas dos componentes do modelo. Esse programa foi desenvolvido em C++. Sendo assim, obteve-se um código-fonte portável. Esse programa foi compilado nos sistemas operacionais Linux e Windows XP®, usando-se nesses sistemas, respectivamente, os compiladores GNU GCC e o Borland C++ Builder®. Para validar dos resultados gerados pelo processador, foi realizada uma comparação de resul-

tados com outro programa, denominado Arena®. Essa validação atingiu resultados satisfatórios, com os quais se pôde ter um ambiente de modelagem e simulação funcional, atendendo às expectativas iniciais da dissertação.

Após esta dissertação ter sido concluída, destaca-se que o uso do *grid* para a execução de simulações em paralelo é válido se a organização que necessita executar uma simulação paralela não possui um *cluster* de computador e participa de um *grid* onde tenha um *cluster* disponível. Caso a organização tenha recursos para montar um *cluster*, recomenda-se que o utilize, pois assim não corre o risco que dados resultantes da simulação sejam colhidos por terceiros.

6.1 Limitações

Esta dissertação apresentou uma ferramenta para o desenvolvimento de modelos de simulação discreta para serem executados em um *grid* computacional. Como se trata de um trabalho inicial, existem algumas limitações nele. É necessário usar programas externos para gerar uma credencial válida para o acesso ao ambiente de execução. Além disso, deve-se usar outros meios para consultar dados referente à disponibilidade, utilização e quantidade de processadores de cada nó do *grid*.

A execução da simulação em vários nós ocorre de modo não automatizado. Isso limita um pouco o uso em vários nós. Em relação à edição dos modelos, a validação dos valores das propriedades está muito precária. O analista deve prestar muita atenção para não errar quando está montando as expressões, pois, se for informado de modo incorreto, a simulação pode resultar em valores de resposta sem lógica, isto é, que não correspondem ao esperado.

6.2 Trabalhos futuros

Ao longo do desenvolvimento desta dissertação novas idéias e melhoramentos iam surgindo. Visando à continuidade deste, segue uma lista de sugestões para trabalhos futuros:

a) melhorar o processador da simulação para que possa suportar características que dêem mais opções ao usuário, como, por exemplo, critério de parada por expressão, tempo de *warm-up*, cálculo de custo, leitura de dados externos, submodelos, animação, suporte a comandos definidos pelo usuário (*script*), validação semântica do modelo (exemplo: para cada *hold*, pelo menos um *release*); melhorar o processador em outros pontos mais específicos, como suporte à simulação de falhas de recursos (tempo de falha, tempo entre falhas e estratégias para a entidade sendo servida, que estão na fila e que chegarão), fila compartilhada, ampliar a análise de expressões (suportando parênteses, funções matemáticas [log, exp, cos, sen, tang, etc.], funções do modelo [tamanho de determinada fila, estado de determinado recurso, etc.] e coleta de estatística); incrementar estratégias de funcionamento da filas (*Last In First Out* [LIFO], menor atributo da entidade, maior atributo da entidade), adicionar transferência de entidade através de seqüências e acrescentar mais componentes;

b) implementar o suporte para multiidioma. Permitir que o usuário escolha o idioma que o Editor utilizará para apresentar os itens de *menu*, os comentários (*hints*), os componentes e suas propriedades, e as mensagens de erro;

c) otimizar a execução dos componentes e diminuir a quantidade de eventos gerados. Atualmente, existem componentes que geram eventos que podem ser descartados. Alguns componentes geram os eventos de saídas com o objetivo de apenas coletar estatísticas e depois mantêm a entidade no fluxo lógico;

d) desenvolver uma versão do processador para executar em paralelo (SRIP). O processador trabalha apenas com o modo MRIP (não automatizado). Alterar o processador para trabalhar com MRIP ou com SRIP daria mais flexibilidade para a ferramenta;

e) melhorar a integração com o *grid*, apresentando mais informações a respeito do ambiente *grid*, como, por exemplo, informações dos nós do *grid* (nome, quantidade de CPUs, dispositivos). Poder realizar esses tipos de consulta, e também poder criar a credencial do *grid* através de comandos internos do Editor, facilitaria o trabalho do analista, por deixar a ferramenta mais completa. Na versão atual, a consulta desses

tipos de informação ocorre com o uso de ferramentas externas; e

f) desenvolver outros tipos de relatórios de saída. O relatório corrente é a captura dos dados impressos no console pela ferramenta. Poderia ser desenvolvida uma opção para gerar esse relatório para arquivo. O formato deste arquivo pode ser bem diversificado; por exemplo, valores separados por vírgula, resultado descrito usando XML, entre outras opções. Esses formatos sempre devem servir para auxiliar a importação dos dados em outros programas ou para ter um formato final de apresentação para que possam ser usados sem a necessidade de um processamento prévio deles em outros programas.

Referências Bibliográficas

[APO 01] APON, A. et al. Cluster computing in the classroom: Topics, guidelines, and experiences. In: THE FIRST IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGRID 2001). [s.n.], 2001. p.476–483.

[ARI 00] ARIEF, L. B.; SPEIRS, N. A. A uml tool for an automatic generation of simulation programs. In: PROCEEDINGS OF THE SECOND INTERNATIONAL WORKSHOP ON SOFTWARE AND PERFORMANCE. ACM Press, 2000. p.71–76.

[BAG 96] BAGRODIA, R. Perils and pitfalls of parallel discrete-event simulation. In: WINTER SIMULATION CONFERENCE. [s.n.], 1996. p.136–143.

[BAN 99] BANKS, J.; CARSON, J. S.; NELSON, B. L. **Discrete–Event System Simulation**. Prentice-Hall, 1999.

[BRA 87] BRATLEY, P.; FOX, B. L.; SCHRAGE, L. E. **A Guide to Simulation**. Springer-Verlag, 1987.

[BRU 00] BRUSCHI, S. M.; SANTANA, R. H. C.; SANTANA, M. J. Asda - an automatic distributed simulation environment. In: 2000 SUMMER COMPUTER SIMULATION CONFERENCE. [s.n.], 2000.

[BRU 03] BRUSCHI, S. M. **ASDA - Um Ambiente de Simulação Distribuída Automático**. USP-São Carlos, 2003. Tese de Doutorado.

[BUM 02] BUMBLE, M.; CORAOR, L. D. An architecture for a nondeterministic distributed simulator. **IEEE Transactions on Vehicular Technology**, [S.l.], v.51, n.3, p.453–471, Maio, 2002.

[BUY 99] Buyya, R., editor. **High Performance Cluster Computing: Architectures and Systems**, v.1. Prentice Hall, 1999.

[CAR 99] CAROTHERS, C. D. et al. Visualizing parallel simulations that execute in network computing environments. **FUTURE GENERATION COMPUTER SYSTEMS**, [S.l.], v.15, n.4, p.513–529, July, 1999.

[CAR 02] CAROTHERS, C. D.; BAUER, D.; PEARCE, S. Ross: A high-performance, low-memory, modular time warp system. **Journal of Parallel Distributed Computing**, [S.l.], p.1648–1669, 2002.

[CHE 02] CHEN, G.; SZYMANSKI, B. K. Lookback:a new way of exploiting parallelism in discrete event simulation. In: 16TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION. [s.n.], 2002. p.138–147.

[DAN 02] DANTAS, M. **Tecnologias de Redes de Comunicação e Computadores**. Axcel Books do Brasil Editora, 2002.

[De 04] De Rose, C. A. F.; NAVAUX, P. O. A. Fundamentos de processamento de alto desempenho. In: ERAD 2004 - 4A. ESCOLA REGIONAL DE ALTO DESEMPENHO. [s.n.], 2004. p.41–66.

[dFF 01] DE FREITAS FILHO, P. J. **Introdução à Modelagem e Simulação de Sistemas**. Visual Books, 2001.

[dR 02] DA ROCHA, F. G.; LULA, J. C.; CABRAL, M. I. C. Componentes de software para a construção de ferramentas de simulação. In: I WORKSHOP EM DESEMPENHO DE SISTEMAS COMPUTACIONAIS E DE COMUNICACÃO. [s.n.], 2002.

[FEN 72] FENG, T. Y. Some characteristics of associative/parallel processing. In: 1972 SAGAMORE COMPUTING CONFERENCE. [s.n.], 1972. p.5–16.

[FER 03] FERREIRA, L. et al. **Introduction to Grid Computing with Globus**. second. ed. IBM Redbooks, 2003.

[FIS 95] FISHWICK, P. A. **Simulation Model Design and Execution: Building digital worlds**. Prentice-Hall, 1995.

[FLY 72] FLYNN, M. J. Some computer organizations and their effectiveness. In: IEEE TRANSACTION ON COMPUTERS. [s.n.], 1972. v.21, p.948–960.

[FOR 03] FORUM, G. G. **Global Grid Forum**. Disponível em: <http://www.gridforum.org/>. Acesso em: Outubro de 2003.

[FOS 99] Foster, I.; Kesselman, C., editors. **The Grid: Blueprint for a New Computing Infrastructure**. Morgan Kaufmann, 1999.

[FOS 01] FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. **International Journal of High Performance Computing Applications**, [S.l.], v.15, n.2, p.200–222, 2001.

[FUJ 99] FUJIMOTO, R. M. Parallel and distributed simulation. In: PROCEEDINGS OF THE 31ST CONFERENCE ON WINTER SIMULATION. ACM Press, 1999. p.122–131.

[FUJ 00] FUJIMOTO, R. M. **Parallel and Distributed Simulation Systems**. Wiley-Interscience, 2000.

[GAM 04] GAMMA, E.; KAISER, W.; QUANTE, J. **JHotDraw**. Disponível em: <http://www.jhotdraw.org/>. Acesso em: Abril de 2004.

[GRI 04] GRIDLAB. **GridSphere: A Portal Framework**. Disponível em: <http://www.gridsphere.org/gridsphere/gridsphere>. Acesso em: Março de 2004.

[HUN 04] HUNTER, J.; MCLAUGHLIN, B. **JDOM project**. Disponível em: <http://www.jdom.org/>. Acesso em: julho de 2004.

[Int 04] International Business Machine. **New to Grid computing**. Disponível em: <http://www-106.ibm.com/developerworks/grid/newto/>. Acesso em: Março de 2004.

[JAC 02] JACOBS, P. H. M.; LANG, N. A.; VERBRAECK, A. D-sol: A distributed java based discrete event simulation architecture. In: 2002 WINTER SIMULATION CONFERENCE. [s.n.], 2002.

[JAC 03] JACOB, B. **Grid computing: What are the key components?** Disponível em: <http://www-106.ibm.com/developerworks/library/gr-overview/>. Acesso em: Julho de 2003.

[KEL 02] KELTON, W. D.; SADOWSKI, R. P.; SADOWSKI, D. A. **Simulation With Arena**. McGraw-Hill, 2002.

[LAW 92] LAW, A. M.; KELTON, W. D. **Simulation Modeling & Analysis**. McGraw-Hill, 1992.

[LCM 04] LCMI. **JARP**. Disponível em: <http://jarp.sf.net/>. Acesso em: Junho de 2004.

[MAC 93] MACCABE, A. B. **Computer Systems: architecture, organization and programming**. New York: McGraw-Hill, 1993.

[MER 03] MEREDITH, M. et al. Exploring beowulf clusters. **J. Comput. Small Coll.**, [S.l.], v.18, n.4, p.268–284, 2003.

[MUR 02] MURPHY, S.; PERERA, T. Successes and failures in uk/us development of simulation. **Simulation Practice and Theory 9**, [S.l.], v.9, p.333–348, 2002.

[NAY 66] NAYLOR, T. H. et al. **Computer Simulation Techniques**. John Wiley & Sons, 1966.

[NS 02] NIEWIADOMSKA-SZYNKIEWICZ, E.; SIKORA, A. Algorithms for distributed simulation - comparative study. In: INTERNATIONAL CONFERENCE ON PARALLEL COMPUTING IN ELECTRICAL ENGINEERING. [s.n.], 2002. p.261–266.

[Obj 03] Object Management Group. **OMG Unified Modeling Language Specification**, March, 2003. version 1.5.

[PAT 03] PATTERSON, D. A.; HENNESSY, J. L. **Arquitetura de Computadores: Uma abordagem quantitativa**. 3. ed. Rio de Janeiro: Campus, 2003. tradução de Vanderberg D. de Souza.

[PIT 03] PITANGA, M. **Computação em Cluster**. Brasport, 2003.

[Rey 88] Reynolds, Jr., P. F. A spectrum of options for parallel simulation. In: PROCEEDINGS OF THE 20TH CONFERENCE ON WINTER SIMULATION. ACM Press, 1988. p.325–332.

[Sci 04] Scientific Discovery through Advanced Computing (SciDAC). **Particle Physics Data Grid**. Disponível em: <http://www.ppdg.net/>. Acesso em: Abril de 2004.

[sit 04a] **The DataGrid Project**. Disponível em: <http://eu-datagrid.web.cern.ch/eu-datagrid/>. Acesso em: Abril de 2004.

[sit 04b] **The Distributed ASCI Supercomputer (DAS)**. Disponível em: <http://www.cs.vu.nl/das/>. Acesso em: Abril de 2004.

[sit 04c] **Grid-Enabled Medical Simulation Services**. Disponível em: <http://www.gemss.de/>. Acesso em: Abril de 2004.

[sit 04d] **NASA's Information Power Grid**. Disponível em: <http://www.ipg.nasa.gov/>. Acesso em: Abril de 2004.

[SKI 88] SKILLICORN, D. B. A taxonomy for computer architectures. **Computer**, [S.l.], v.21, n.11, p.46–57, 1988.

[STE 01] STERLING, T. An introduction to pc clusters for high performance computing. **International Journal of High Performance Computing Applications**, [S.l.], v.15, n.2, p.92–101, May, 2001.

[SUL 03] SULISTIO, A.; YEO, C. S.; BUYYA, R. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. **International Journal of Software: Practice and Experience**, [S.l.], 2003.

[Sun 03] Sun Microsystems. **Java technology**. Disponível em: <http://java.sun.com/>. Acesso em: julho de 2003.

[The 04] The Apache Software Foundation. **Xerces-C++ Parser**. Disponível em: <http://xml.apache.org/xerces-c/>. Acesso em: julho de 2004.

[TUE 03] TUECKE, S. et al. Open grid services infrastructure (ogsi): Version 1.0. Global Grid Forum, 2003. Relatório técnico.

[UNG 03] UNGER, J.; HAYNOS, M. **A visual tour of Open Grid Services Architecture**. Disponível em: <http://www-106.ibm.com/developerworks/grid/library/gr-visual/>. Acesso em: Novembro de 2003.

[Uni 04] University of Chicago. **The Globus Toolkit**. Disponível em: <http://www-unix.globus.org/toolkit/>. Acesso em: Fevereiro de 2004.

[VAD 03] VADHIYAR, S. S.; DONGARRA, J. J. A performance oriented migration framework for the grid. In: 3RD IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID. [s.n.], 2003. p.130–137.

[VAU 03] VAUGHAN, F. A.; GROVE, D. A.; CODDINGTON, P. D. Communication performance issues for two cluster computers. In: PROCEEDINGS OF THE TWENTY-SIXTH AUSTRALASIAN COMPUTER SCIENCE CONFERENCE ON CONFERENCE IN RESEARCH AND PRACTICE IN INFORMATION TECHNOLOGY. Australian Computer Society, Inc., 2003. p.171–180.

[vdS 03] VAN DER STEEN, A. J. An evaluation of some beowulf clusters. **Cluster Computing**, [S.l.], v.6, n.4, p.287–297, October, 2003.

[vL 01] VON LASZEWSKI, G. et al. A java commodity grid kit. **Concurrency and Computation: Practice and Experience**, [S.l.], v.13, n.8-9, p.643–662, 2001.

[ZEI 00] ZEIGLER, B. P.; PRAEHOFER, H.; KIM, T. G. **Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems**. second. ed. Harcourt India Private Limited, 2000.

Apêndice A

Definição DTD do arquivo de armazenamento de modelos utilizado pelo GridSimulator

```
<!ELEMENT gridsimeditor ( version, drawing, componentslist, commentslist, componentsflow ) >

<!ELEMENT version EMPTY >
<!ATTLIST version value NMTOKEN #REQUIRED >

<!ELEMENT drawing ( class, variables ) >
<!ATTLIST drawing executiontime NMTOKEN #REQUIRED >
<!ATTLIST drawing executiontimeunit NMTOKEN #REQUIRED >
<!ATTLIST drawing generatedefaultstats ( false | true )#REQUIRED >
<!ATTLIST drawing replicationnumber NMTOKEN #REQUIRED >

<!ELEMENT class ( #PCDATA ) >

<!ELEMENT variables ( variable* ) >

<!ELEMENT variable EMPTY >
<!ATTLIST variable name CDATA #REQUIRED >
<!ATTLIST variable value CDATA #REQUIRED >

<!ELEMENT componentslist ( component+ ) >


<!ELEMENT component ( fixedattribute | userattribute )+ >
<!ATTLIST component id NMTOKEN #REQUIRED >
```

```
<!ATTLIST component type NMTOKEN #REQUIRED >
<!ATTLIST component visible ( false | true ) #REQUIRED >
<!ATTLIST component xpos NMTOKEN #REQUIRED >
<!ATTLIST component ypos NMTOKEN #REQUIRED >

<!ELEMENT fixedattribute ( value? ) >
<!ATTLIST fixedattribute class CDATA #REQUIRED >
<!ATTLIST fixedattribute key NMTOKEN #REQUIRED >
<!ATTLIST fixedattribute label CDATA #REQUIRED >
<!ATTLIST fixedattribute value CDATA #REQUIRED >
<!ATTLIST fixedattribute visible ( false | true ) #REQUIRED >

<!ELEMENT value ( #PCDATA ) >

<!ELEMENT userattribute EMPTY >
<!ATTLIST userattribute class CDATA #REQUIRED >
<!ATTLIST userattribute key NMTOKEN #REQUIRED >
<!ATTLIST userattribute label CDATA #REQUIRED >
<!ATTLIST userattribute value CDATA #REQUIRED >
<!ATTLIST userattribute visible ( false | true ) #REQUIRED >

<!ELEMENT commentslist ( comment* ) >

<!ELEMENT comment ( value ) >
<!ATTLIST comment height NMTOKEN #REQUIRED >
<!ATTLIST comment id NMTOKEN #REQUIRED >
<!ATTLIST comment type NMTOKEN #REQUIRED >
<!ATTLIST comment width NMTOKEN #REQUIRED >
<!ATTLIST comment xpos NMTOKEN #REQUIRED >
<!ATTLIST comment ypos NMTOKEN #REQUIRED >

<!ELEMENT componentsflow ( flow* ) >

<!ELEMENT flow EMPTY >
<!ATTLIST flow fromId NMTOKEN #REQUIRED >
<!ATTLIST flow kind ( false | true ) #REQUIRED >
<!ATTLIST flow toId NMTOKEN #REQUIRED >
<!ATTLIST flow visible ( false | true ) #REQUIRED >
```

Apêndice B

Diagrama de seqüência de envio do modelo para executar no grid

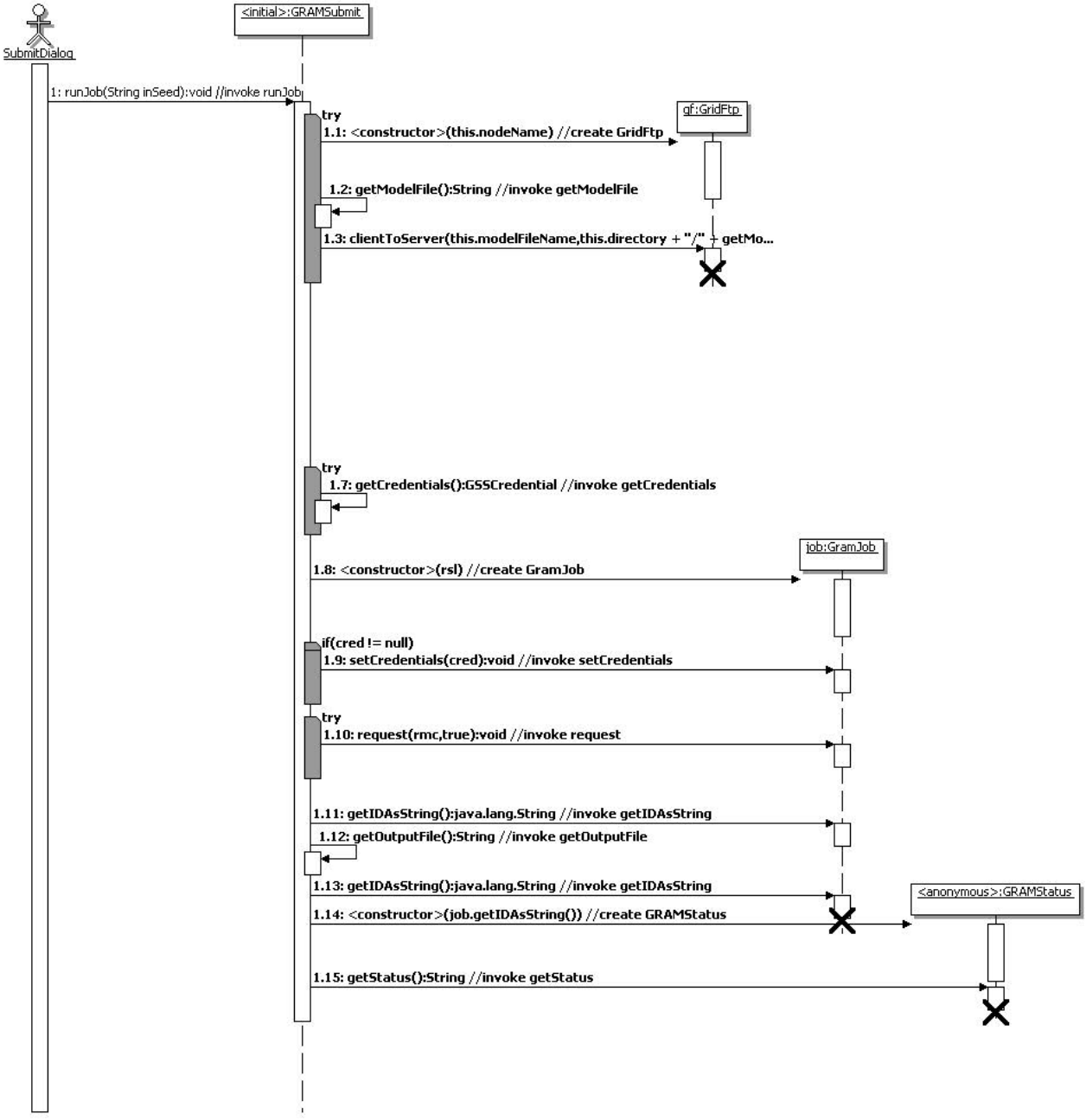

Figura B.1: Diagrama de seqüência

Apêndice C

Gráficos de comparação de perfis

Neste apêndice encontram-se os gráficos de comparação dos perfis das variáveis analisadas. Esses gráficos apresentam uma comparação visual do comportamento das variáveis nos dois programas. Para identificar as variáveis foi utilizada a seguinte notação: "LLVnCn", onde LL pode ser "AR", que informa que os dados foram gerados pelo Arena®, ou "GS", que informa que os dados foram gerados pelo GridSimulator. E n identifica a variável e a configuração apresentadas na seção 5.3.1. A seguir estão apresentados os perfis da figura C.1 até a C.16. Os pontos marcados com círculo foram gerados pelo GridSimulator, e os marcados com quadrado foram gerados pelo Arena®.

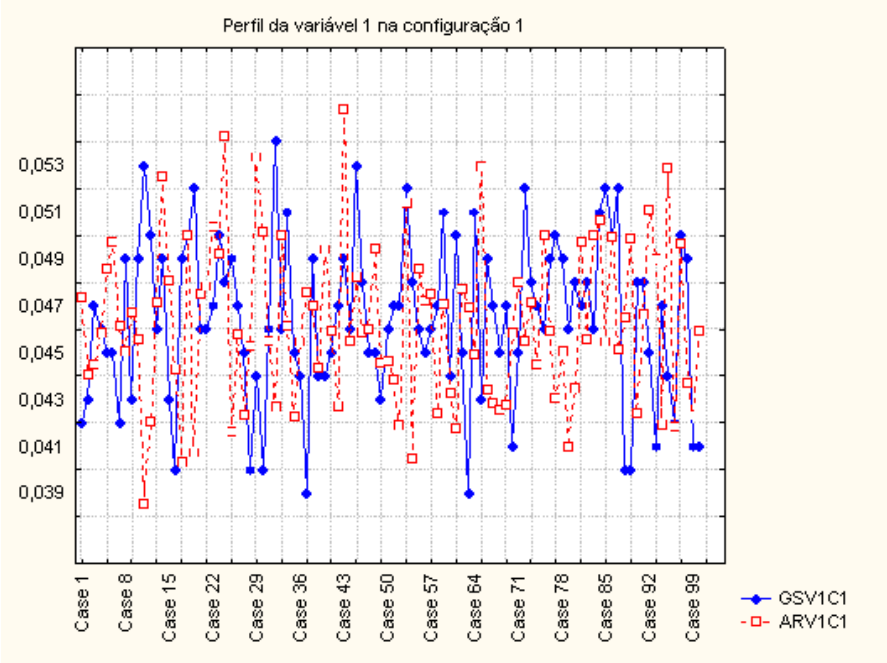

Figura C.1: Perfil da variável 1 na configuração 1

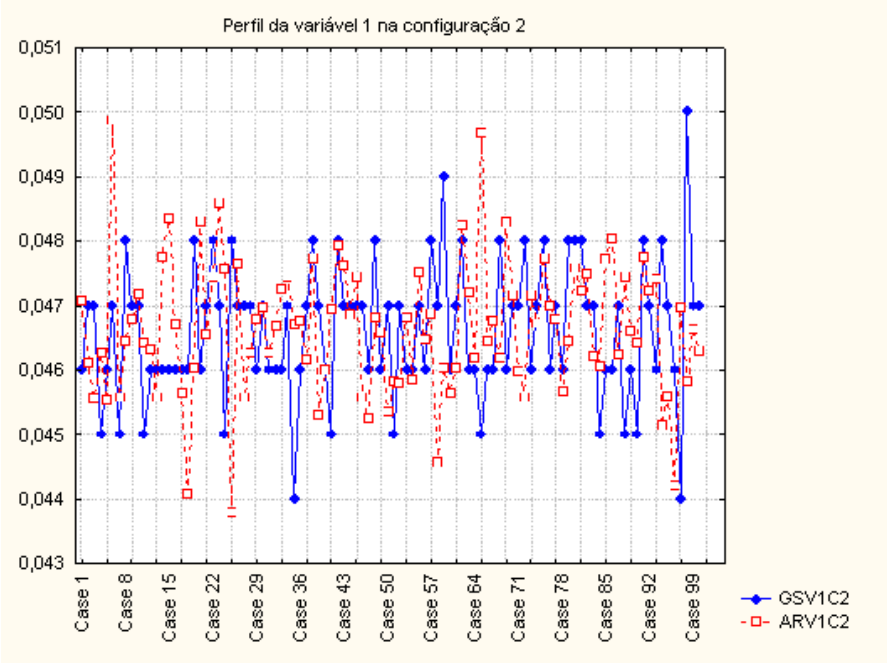

Figura C.2: Perfil da variável 1 na configuração 2

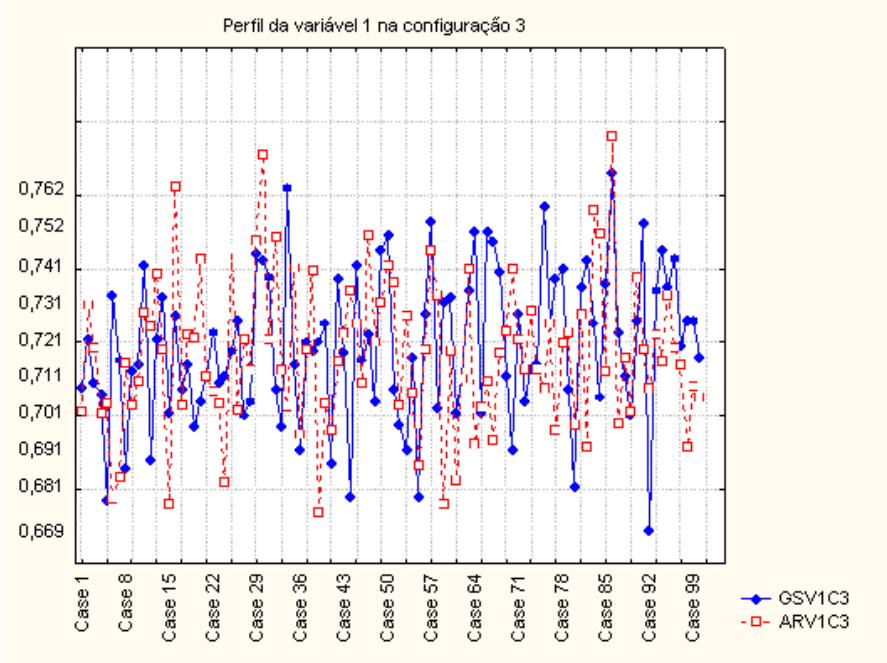

Figura C.3: Perfil da variável 1 na configuração 3

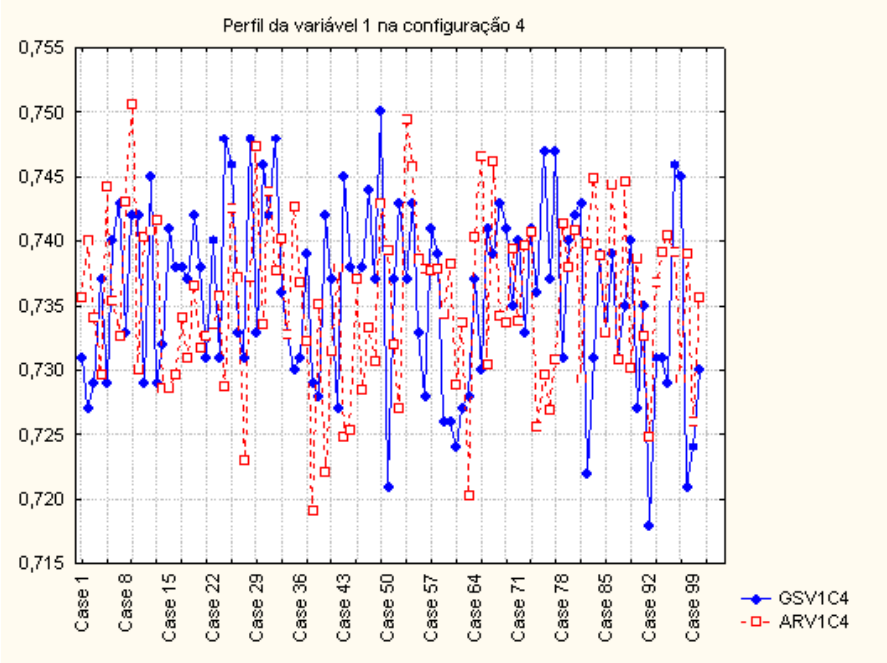

Figura C.4: Perfil da variável 1 na configuração 4

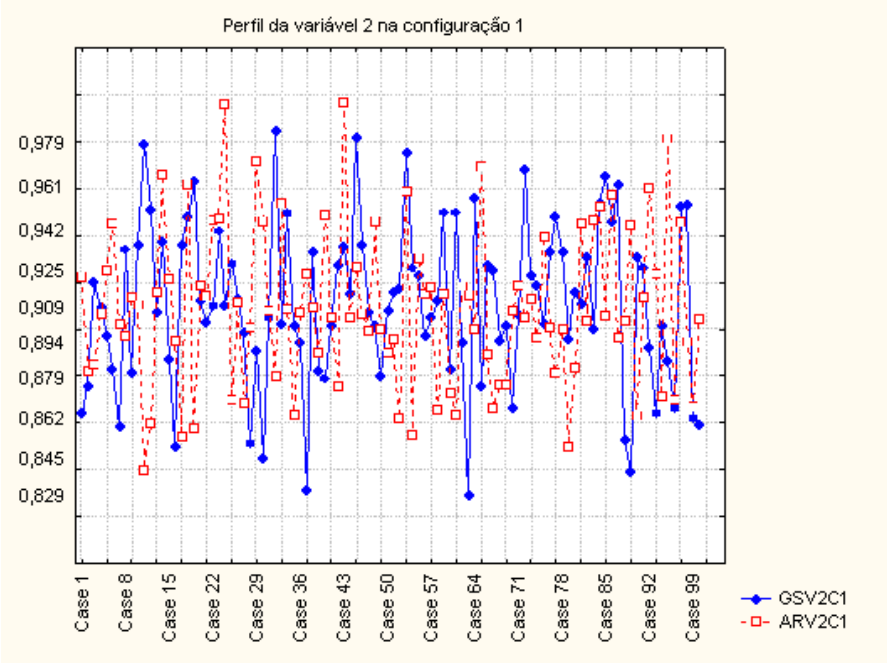

Figura C.5: Perfil da variável 2 na configuração 1

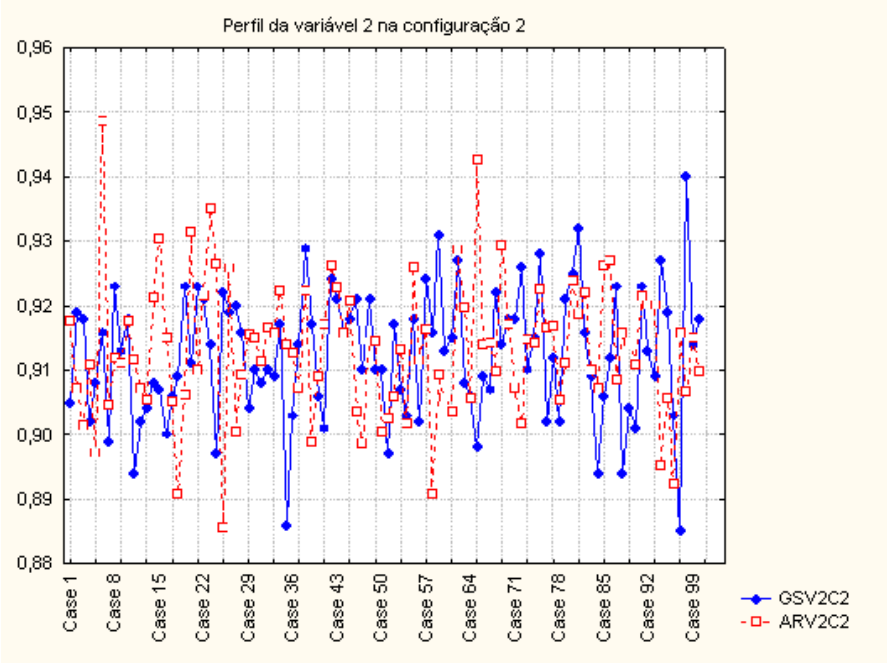

Figura C.6: Perfil da variável 2 na configuração 2

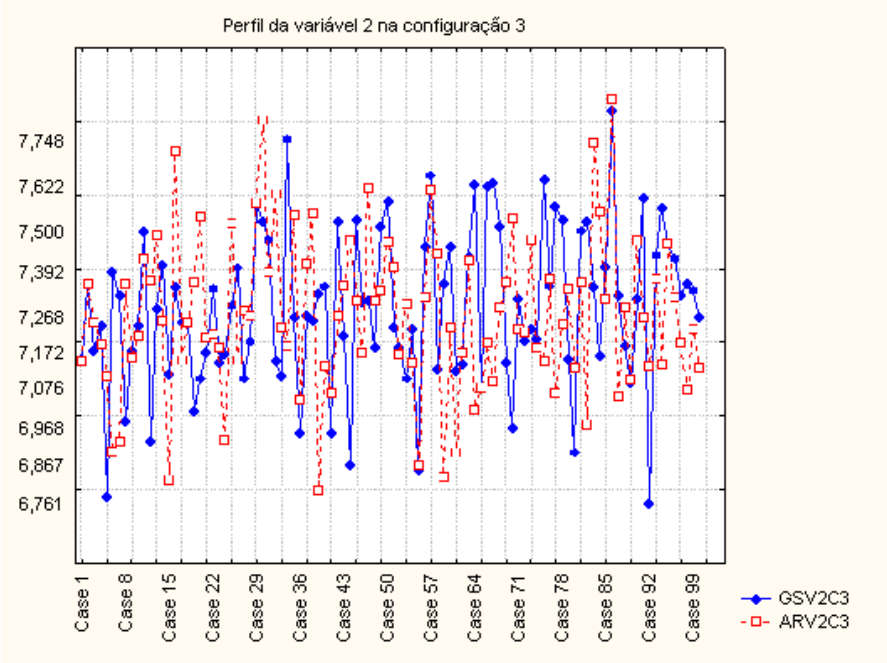

Figura C.7: Perfil da variável 2 na configuração 3

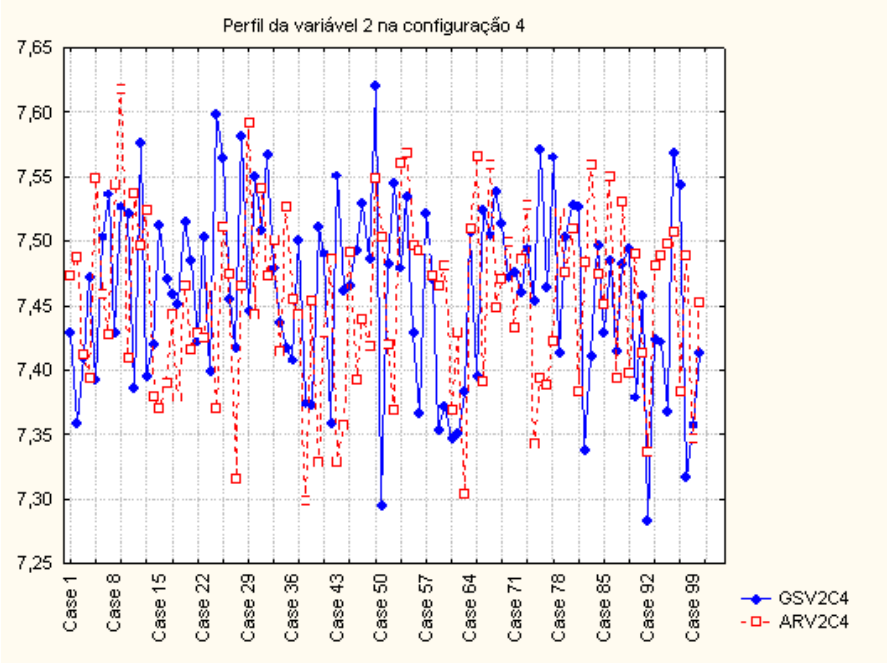

Figura C.8: Perfil da variável 2 na configuração 4

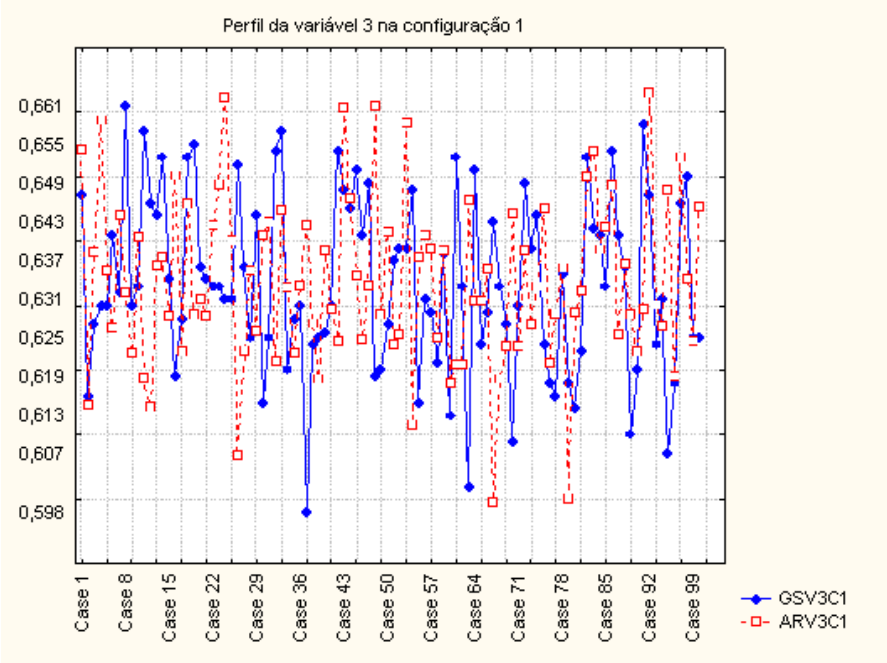

Figura C.9: Perfil da variável 3 na configuração 1

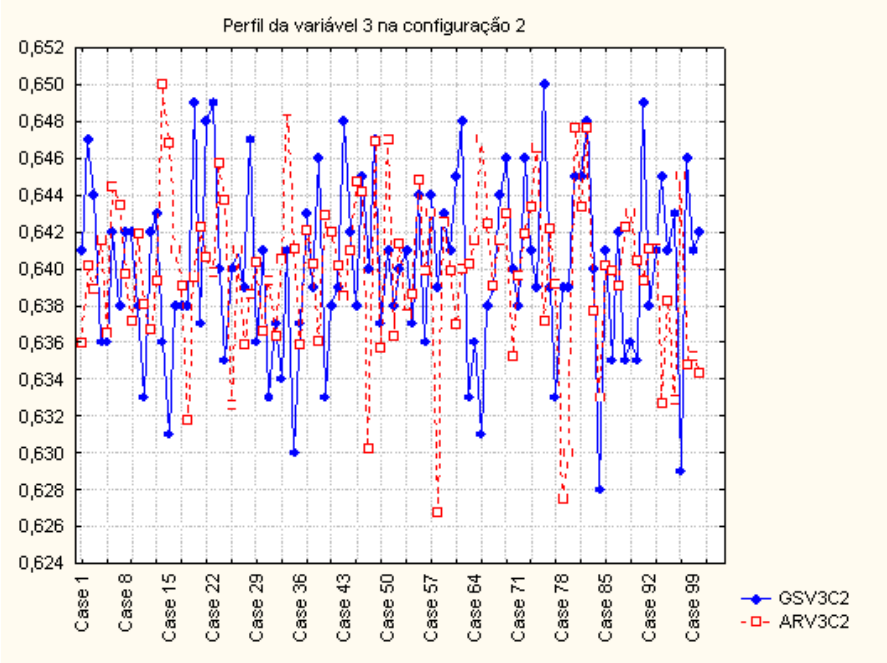

Figura C.10: Perfil da variável 3 na configuração 2

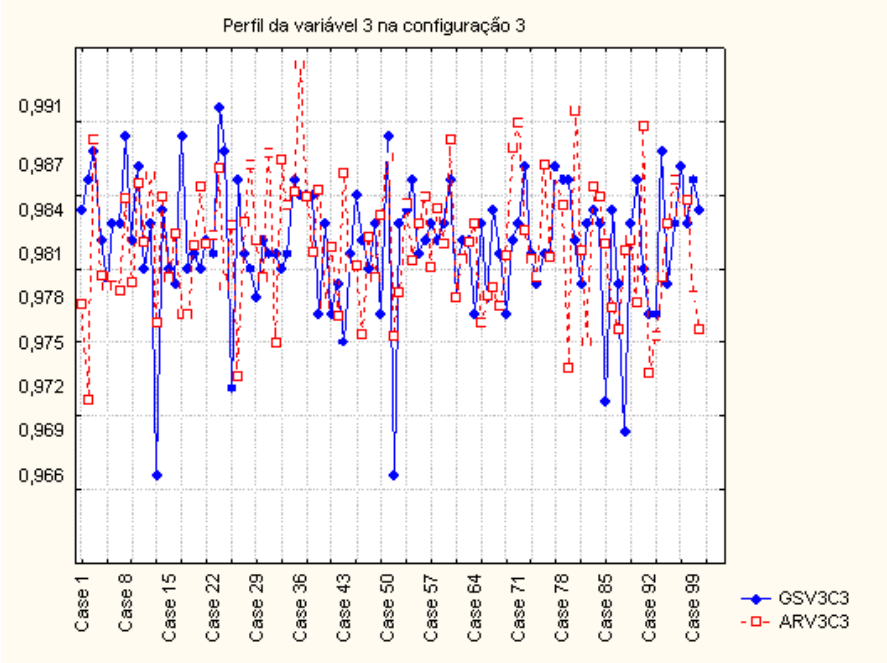

Figura C.11: Perfil da variável 3 na configuração 3

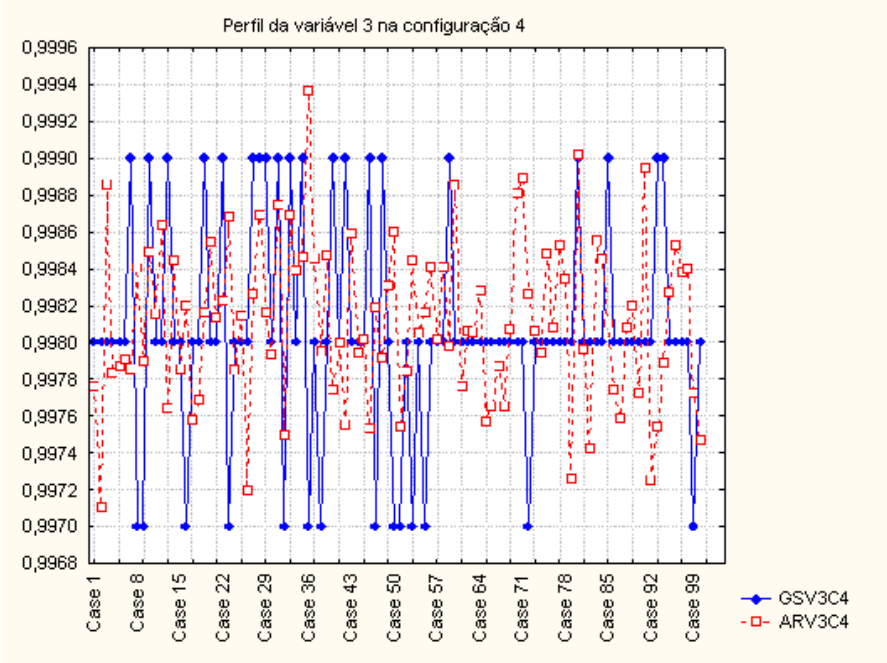

Figura C.12: Perfil da variável 3 na configuração 4

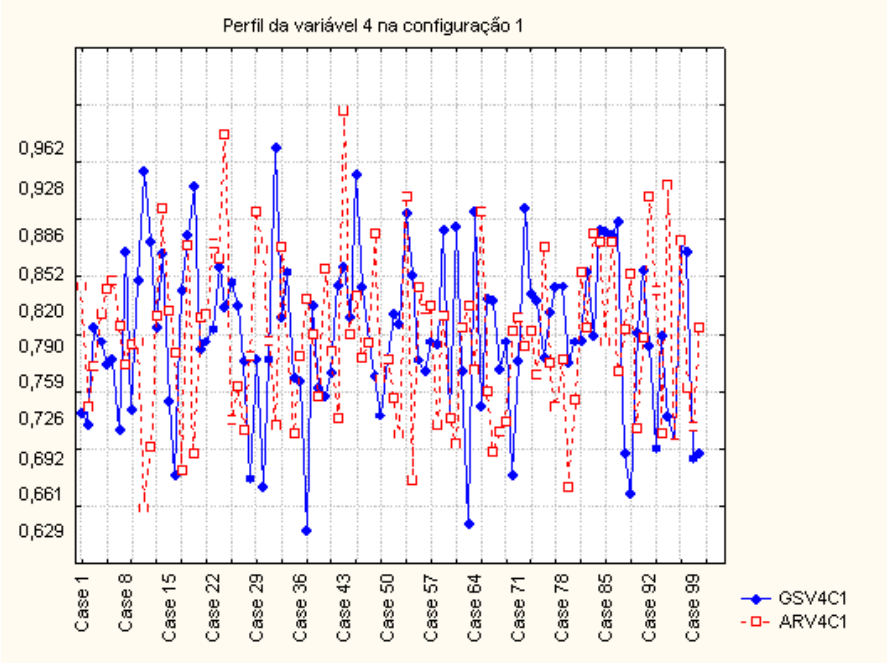

Figura C.13: Perfil da variável 4 na configuração 1

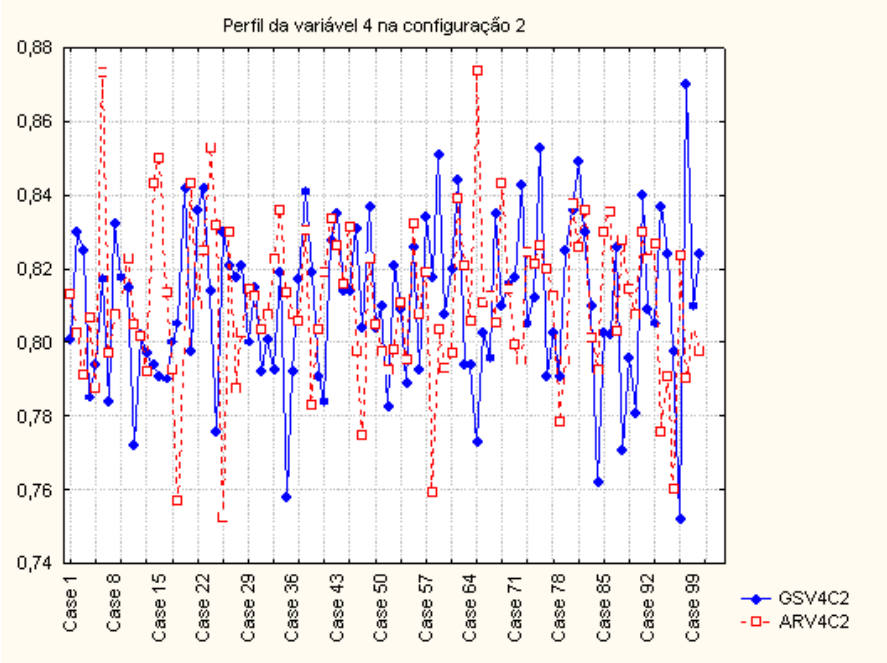

Figura C.14: Perfil da variável 4 na configuração 2

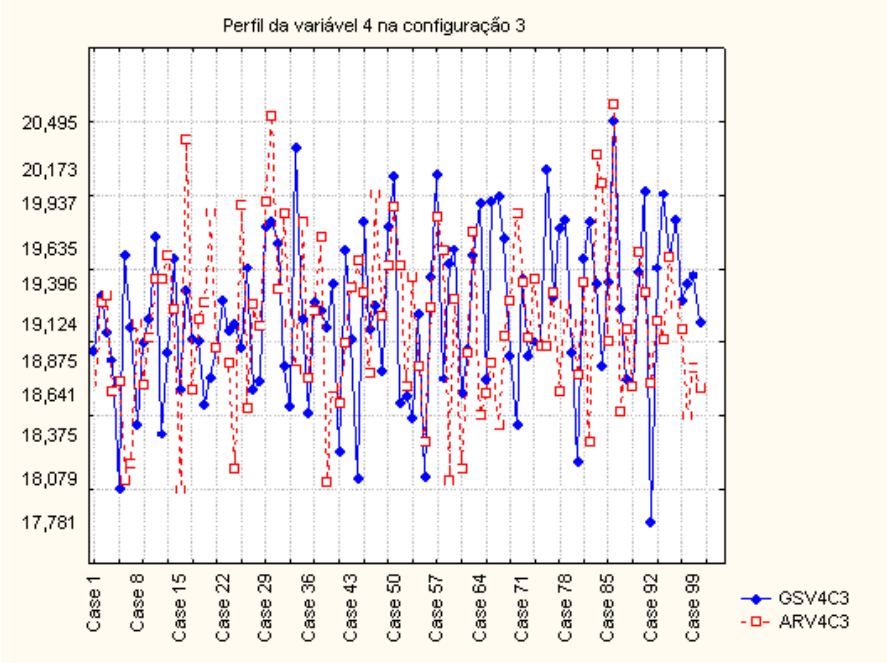

Figura C.15: Perfil da variável 4 na configuração 3

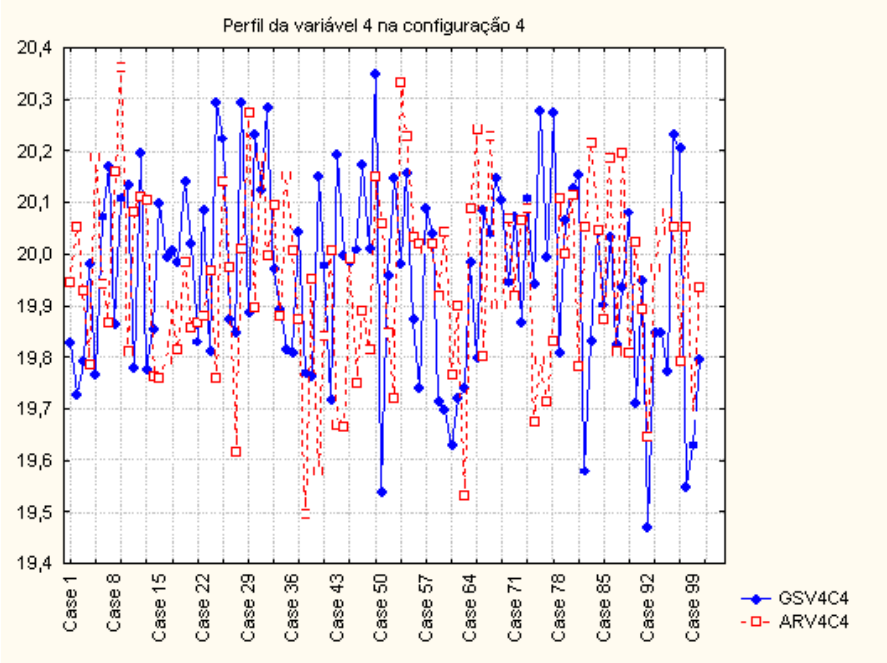

Figura C.16: Perfil da variável 4 na configuração 4